**Git**

Gli oggetti tree definiscono la struttura ad albero delle versioni del progetto, che è costituito dalla directory iniziale e da varie subdirectory.  
Gli oggetti commit indicano l’evoluzione del progetto. Un nuovo oggetto commit punta alla radice del tree del progetto e ai commit immediatamente precedenti. Se sono due o più, si tratta di un merge.  
Il file staging area contiene l’elenco dei file che saranno aggiunti nella nuova versione con commit.  
Il file HEAD indica il ramo corrente, inizialmente il master. Tag associa un tag all’oggetto commit corrente. Il comando branch produce un nuovo ramo che punta al commit corrente, che è il commit puntato dal ramo corrente. Con il comando checkout il nuovo ramo diventa quello corrente (è copiato nella working directoru), HEAD punterà al nuovo ramo. Con un checkout del master, la working directory torna a contenere la copia della versione corrente del master. Conflitto: se il contenuto di un file della working directory e quello del file omonimo del ramo da fondere non sono componibili automaticamente: l’utente deve risolvere i conflitti manualmente.

1. Il punto di partenza della pipeline di un sistema di continuous integration può essere un ambiente di version control come Git. (V)
2. Nel repository di Git un oggetto commit può essere collegato ad un altro commit al più. (F)
3. In Git il comando commit esporta un progetto locale nel repository remoto (F)

**Project management: CPM e WBS**

La durata di un task dipende dalle risorse, non dall’impegno. L’albero WBS non si usa per analisi del percorso critico, bensì il CPM dove il percorso critico è quello più lungo. I task sul percorso critico non possono essere ritardati senza variare la durata complessiva, gli altri task non-critici sì e il margine di ritardo è detto slack time.

1. La durata di un task in un diagramma CPM è stabilita in funzione dell’impegno e delle risorse coinvolte (F) 🡪 è logico se ci pensi, prescinde dall’impegno
2. La durata di un task in un diagramma PERT/CPM prescinde dall’impegno effettivo richiesto agli esecutori del task. (V) 🡪 stessa domanda di sopra
3. Una struttura WBS (work break structure) nel project management è utile per l’analisi del percorso critico (F) 🡪 Perché qui abbiamo una struttura ad esempio ad albero, delle attività da fare prima che siano messe con delle precedenze
4. Un diagramma CPM si usa nel project management per l’analisi del percorso critico (V)

**Grafi marcati, free-choice e reti di Petri**

Macchine a stati: ogni transazione ha 1 input e 1 output. Esistono le scelte libere. No fork e join. La marcatura iniziale ha 1 token.

Grafo marcato: ogni posto può avere 1 input e 1 output. Fork e join. No conflitti e confluenze. Bounded se 1 token in ogni circuito (iff fort. connesso). Live se ogni circuito ha almeno 1 token. Un circuito è marcato se il suo token count > 0. Il #max\_token in un posto è il min token count fra i circuiti di cui fa parte. Safe se per ogni posto si può trovare un circuito (tra quelli di cui fa parte) che contiene 1 solo token.

FC: sia posti sia transizioni possono avere #input>1 e #output>1. Ciascun posto con #output>1 è l’unico posto di input di quelle specifiche transizioni. Sifone: quando esce non torna più. Trappola: una volta che entra, non lo perde. T Kommoner: live se ogni sifone contiene (o è uguale a) una trap. m. i. (perché il sifone così conterrà sempre token). Live 🡪 DF (tutte le trans possono scattare). Se ha un deadlock, non è live. Unbounded 🡪 potrebbe essere DF ma non live.

AC: Kommoner è sufficiente (ma non necessario). La rete è live se:  
- i sifoni contengono trap. m. i. (ma se non le contengono, potrebbe lo stesso essere live)  
OPPURE  
- un sifone potrebbe essere controllato da un place invariant.

Un place invariant è una combinazione lineare di posti che dà lo stesso numero di token per qualunque marcatura. Un place invariant garantisce che la marcatura (#token) di un sifone sia sempre > 0. Se un sifone contiene un invariante, è controllato da quell’invariante.

Due posti formano un invariante se sommando i token che hanno otteniamo sempre lo stesso numero di token qualunque sia la marcatura (es. 1). Lo si vede anche dal grafo delle marcature: qualsiasi marcatura, viene sempre 0+1.

1. Le confluenze si trovano nelle reti di tipo State Machine e Grafi Marcati. (F) 🡪 Non ci sono confluenze nei grafi marcati
2. In un grafo marcato tutte le transizioni devono avere un numero di posti di input e/o di output > 1 (F)
3. Nei grafi marcati un circuito si dice marcato se tutti i suoi posti contengono almeno un token. (F) 🡪 un circuito è marcato se il token count è > 0, non per forza un token per ogni posto
4. Per determinare il percorso critico in un progetto si può usare la tecnica di calcolo del tempo ciclo nei grafi marcati temporizzati (V)
5. Se un grafo marcato connesso ma non strettamente connesso è live, allora non è necessariamente unbounded. (F) 🡪 bounded se e solo se è strettamente connesso, quindi se è semplicemente connesso è unbounded a prescindere dalla liveness
6. Un grafo marcato può essere analizzato mediante trappole e sifoni; in questo caso i circuiti corrispondono a trappole. (F) 🡪 non c’entra niente, quelle sono le FC e le AC
7. Una rete free-choice bounded e fortemente connessa è sicuramente live. (F)
8. Una rete free-choice fortemente connessa i cui task sono passanti o fork è unbounded (V)
9. Il pattern free choice non si trova soltanto nelle reti di categoria free choice. (V)
10. Una rete di Petri bounded è live se (e soltanto se) il grafo delle marcature raggiungibili (RG) è fortemente connesso (F)
11. POSTI IN SERIE: Due posti, p1 e p2, di una rete di Petri si dicono in serie se c’è una transizione t che ha p1 come unico input e p2 come unico output. (F) 🡪 Non è detto che t sia l’unica transizione ad avere p1 come unico input; ci potrebbe essere anche una t2 da p1 a p3.
12. TRANSIZIONI IN SERIE: Due transizioni, t1 e t2, di una rete di Petri si dicono in serie se c’è un posto che ha t1 come unico input e t2 come unico output. (F) 🡪 Perché il posto deve essere l’unico input di t2, o meglio deve abilitarla.
13. TRANSIZIONI IN PARALLELO: In una rete di Petri due transizioni si dicono in parallelo se hanno un solo posto di input e tale posto è comune ad entrambe (F) 🡪 Devono avere in comune anche l’output
14. La regola di riduzione che riguarda le transizioni in parallelo si applica quando due o più transizioni hanno un posto di input in comune e un posto di output in comune. (F) 🡪 Occorre aggiungere che le transizioni non devono avere altri posti di input (output) oltre a quello in comune.
15. Una rete di Petri in cui tutti gli elementi (posti e trans) hanno almeno 1 input e almeno 1 output è anche fortemente connessa. (F) 🡪 fortemente connessa significa che da ogni posto bisogna potersi muovere in tutti i posti della rete (pensa all’esempio dove posso passare da sx a dx ma non posso tornare indietro)
16. Nelle reti di Petri è possibile che il numero complessivo dei token presenti nei posti di un sifone non sia mai zero anche se il sifone non contiene alcuna trappola. (V) 🡪 Nelle reti AC un sifone può essere controllato da un invariante (place invariant)
17. Nelle reti AC il numero dei token presenti nei posti di una trappola non è mai zero (F) 🡪 basta che il token venga imprigionato, non è necessario che ci sia un token per ogni posto
18. Applicando le regole di riduzioni di una rete di Petri, la rete ridotta potrebbe appartenere ad una sottoclasse più semplice di quella della rete originaria. (V) 🡪 La prima rete di Petri che abbiamo visto (AC) che però con le riduzioni si riduce a una rete di livello più semplice.

**Modello dataflow (DFD)**

Attori e datastore possono essere collegati solo con attività (che sono elementi attivi che “consumano”).

1. In un modello dataflow (DFD) un attore esterno non può essere collegato ad un’attività composta. (F)
2. In un modello dataflow (DFD) un’attività composta non può essere collegata direttamente ad un attore esterno. (F) 🡪 uguale alla precedente, nel diagramma di contesto dei DFD l’attività di top-level è composta ed è collegata agli attori esterni.
3. In un modello dataflow (DFD) un attore esterno può ricevere dati direttamente da un datastore. (F) 🡪 uguale alla precedente, attori esterni e datastore possono essere collegati solo con le attività, che sono gli unici elementi attivi, ovvero che possono “consumare”
4. Un modello dataflow (DFD) è composto soltanto da attori esterni, da attività e da collegamenti che trasportano i dati da un attore ad un’attività, da un’attività ad un attore o da un’attività ad un’altra attività. (F) 🡪 mancano all’appello i datastore, che immagazzinano dati che possono essere letti in qualsiasi ordine

**Testing white box (WBT)**

(a&&b) || c 🡪 5 true, 3 false (cT e 4 combinazioni di a e b (=4) + cF, aT, bT (=1), totale 5)  
(a||b) && c 🡪 3 true, 5 false (cT e 3 combinazioni di a e b (esclusa la quarta che era aF, bF))

1. Nel testing white box (WBT) il numero dei casi di test necessari a coprire una condizione tripla contenente un operatore or (||) e un operatore and (&&) dà un numero di risultati true maggiore di quello dei risultati false indipendetemente dalle parentesi () introdotte. (F) 🡪 dimostrato sopra, le parentesi cambiano tutto
2. Nel testing white box (WBT) di un programma è possibile che il numero minimo di test necessari a coprire le condizioni multiple sia > quello dei test necessari a coprire i percorsi. (V)
3. Nel testing white box il numero minimo di casi di test necessari per la copertura delle condizioni multiple è sempre > quello per la copertura dei link. (F)
4. Nel testing white box il numero minimo di casi di test necessari per la copertura dei link (branch o edge) di un programma può essere uguale a quello per la copertura dei percorsi. (V)
5. Nel testing white box il n. minimo di casi di test necessari per la copertura dei nodi di un programma è sempre < quelli per la copertura dei link (branch o edge). (F) 🡪 possono coincidere se non si è lasciato scoperto alcun link
6. In un grafo WBT che, oltre ai nodi d’azione, contiene tre condizioni in cascata – la prima è la tripla x || y || z, la seconda è la doppia a&&b e la terza è la doppia c&&d – il numero minimo di test per la copertura dei percorsi è uguale al minimo numero di test per la copertura delle condizioni multiple. (V: le tre condizioni sono sequenziali: 8 casi per la tripla (CM), 8 percorsi per le 3 condizioni sequenziali). 🡪 …. E se hanno delle correlazioni… boh ….
7. In un grafo WBT che oltre ai nodi d’azione contiene 2 condizioni, la doppia a&&b e la semplice x>10, il numero minimo di test per la copertura dei percorsi può essere uguale al minimo numero di test per la copertura delle CM. (V e F) 🡪 Falso si considerano i test, vero se si considerano i numeri dei test; valide entrambe le risposte Falso e Vero.

**Sviluppo agile (+ SCRUM)**

- Individuals and interactions **over** *process and tools*  
- Working software **over** *comprehensive documentation*  
- Customer collaboration **over** *contract negotiation*  
- Responding to change **over** *following a plan*

- Customer satisfaction by early and continuous frequent delivery of valuable software  
- Welcome changing requirements  
- Close daily cooperation business people – developers  
- Projects are built around motivated individuals who should be trusted  
- Face-to-face conversation  
- Working software = measure of progress  
- Sustainable development, constant pace  
- Attention to technical excellence and good design  
- Simplicity  
- Best architectures, requirements and designs emerge from self-organizing teams  
- Regularly team reflects on how to be more effective (and adjusts accordingly)

Extreme programming (XP):  
- **sviluppo agile con pianificazione settimanale** (**planning game**) 🡪 release planning e iteration planning. Il primo è la raccolta dei requisiti basati su user stories, analisi costi benefici e scelta dei requisiti da includere nella release. Il secondo definizione dei task, allocazione dei task agli sviluppatori e implementazione con test-driven development.  
- **user stories**: molto semplici, gli utenti (committente) scrivono le loro esigenze/requisiti. Gli use cases sono più complessi perché descrivono interazioni utente-sistema  
- **pair programming**: si lavora in 2 sullo stesso programma, benefici: coesione di gruppo e on-the-job training per i neoassunti, conoscenza distribuita, review informale ma continua  
- **test driven development**: si scrivono i test del codice per capire i requisiti di ciò che si deve implementare. I test servono per il regression testing dopo l’introduzione di nuove funzionalità.  
- **continuous integration**: integrare il lavoro quotidiano degli sviluppatori in una baseline condivisa. Comporta l’esecuzione dei test di integrazione (svolti anche da esterni). Obiettivo sistema sempre coerente grazie a small release controllate.  
- **refactoring**: il principio “design for change” non è essenziale perché è difficile e rischioso anticipare i cambiamenti futuri. Ma un componente può diventare “disordinato”, in tal caso si può renderlo più leggibile e strutturato e agevolare estensioni future.

1. Nel manifesto dello sviluppo agile i processi sono più importanti delle persone. (F) 🡪 interazione individuale in primo piano, libertà al programmatore
2. Nello sviluppo agile attenersi al piano prestabilito è considerato un principio assoluto. (F) 🡪 possono essere cambiati i requisiti durante lo sviluppo, “responding to change over following a plan”
3. Lo sviluppo agile del software si propone di contrastare attivamente l’aumento della complessità dei programmi. (secondo me F)
4. Lo sviluppo agile si può ritenere un’alternativa completa al modello unificato di sviluppo del software. (F) 🡪 Non lo è perché il modello unificato parte da una visione di insieme che non sempre c’è nello sviluppo agile
5. Il modello unificato del ciclo di vita del software include il modello incrementale e quello evolutivo ma non quello waterfall. (F)
6. Nello sviluppo agile, mentre si scrive il codice della versione corrente si cerca già di tenere in conto i cambiamenti futuri più probabili. (F) 🡪 Non c’è certezza di arrivare a dover fare dei cambiamenti, quindi si può aspettare di farlo più avanti, magari se c’è un po’ di entropia quando c’è tempo
7. Nello sviluppo agile per integrazione continua si intende la presa in carico di nuovi requisiti nell’elaborazione della versione corrente. (F) 🡪 lo sviluppo agile ha reso l’integrazione un fatto quotidiano: gli elementi da controllare sono pochi perché si riferiscono al lavoro di un giorno. Significa integrare giornalmente il lavoro degli sviluppatori in una baseline condivisa
8. Il team di sviluppo e il team delle operations si occupano entrambi dell’integrazione continua. (F: l’operations team si occupa del continuous delivery). (F) 🡪 L’operations team si occupa del continuous delivery
9. Nella programmazione estrema il refactoring si riferisce alla pianificazione incrementale dello sviluppo del software (F)
10. Il pair programming può facilitare il training on the job dei neoassunti. (V)
11. La proprietà collettiva è un obiettivo della programmazione estrema.
12. Le user stories sono meno dettagliate degli use cases. (V)
13. Un’istanza di processo SCRUM è governata dallo SCRUM master. (F) 🡪 SCRUM si usa per gestire prodotti software complessi. Il prodotto evolve per incrementi in fasi dette **sprint** (durata standard: 1 mese). Lo sprint comprende:  
    - Sprint Planning: determina elementi del product backlog, definisce il goal dello sprint. Pianifica il lavoro dei primi giorni in unità giornaliere (o inferiori).  
    - Daily Scrum: ogni membro sintetizza il lavoro del giorno prec. e illustra quello del giorno corrente  
    - Sprint Review: a fine mese si presentano l’incremento fatto e le carenze. Si discute dei problemi e delle soluzioni. Alla fine si aggiorna il product backlog  
    - Sprint Retrospective: a fine progetto si analizza come è stato fatto il lavoro e si pianificano miglioramenti

L’obiettivo è di sviluppare un incremento del prodotto potenzialmente rilasciabile. I risultati parziali sono potenzialmente rilasciabili. Il product backlog è l’elenco di requisiti (può crescere). Sprint backlog = elementi product backlog selezionati per lo sprint + piano dello sprint. I membri del team devono avere una comprensione condivisa di ciò che si intende per lavoro fatto (completo). La definizione di lavoro fatto (DONE) può evolvere nel tempo.  
Un team SCRUM è formato da product owner (gestisce in maniera esclusiva il product backlog, stabilisce l’ordine dei requisiti), team di sviluppo e Scrum master (assicura che i principi di SCRUM siano compresi e fa da coaching agli altri due). Principi SCRUM: trasparenza, ispezione, adattamento.

**DevOps**

Obiettivo: integrare attività e team di sviluppo e di messa in produzione di un software.  
- Development team: sviluppa il software  
- Operations team: lo integra nell’ambiente di produzione, mette il cliente in grado di usarlo. Si occupa del provisioning, della gestione dell’infrastruttura e degli strumenti di lavoro  
L’integrazione dei 2 team deve essere efficiente e si ottiene con l’istituzione di team multifunzionali. Supporto tecnologia per automatizzare alcuni processi. DevOps integra:  
- continuous integration (CI): lavoro giornaliero, può essere controllata con test automatizzati (anche ad ogni nuovo rilascio). Test eseguiti in sequenza, il processo è una pipeline di attività: se un’attività produce un risultato positivo, si esegue la succ., altrimenti sviluppatori devono apportare modifiche e far rieseguire l’attività.  
- continuous delivery (CD): eseguibile trasferito in staging environment (che riproduce l’ambiente di produzione). Si fanno test, tra cui performance, controllo qualità prodotto prima della produzione. Infine deployment nell’ambiente di produzione.  
Benefici automazione pipeline: qualità (controlli freq), flessibilità sviluppo (agile), integrazione team, efficienza.

1. L’uso dei microservizi può essere una valida risposta all’esigenza di rendere i sistemi monolitici più facili da mantenere (V)
2. I microservizi possono essere rilasciati separatamente perché hanno un buon grado di autonomia. (V)

**Metodo di Fagan**

1. Il metodo di Fagan è un metodo per la validazione del software. (F) 🡪 E’ un metodo per la verifica del software. La validazione invece è quello che ci permette di fare il check finale da parte del committente prima che riceva il prodotto.
2. Nel metodo di Fagan gli ispettori possono suggerire all’autore dei miglioramenti progettuali. (F) 🡪 no solution hunting, no design alternatives. Gli ispettori evidenziano solo i difetti.
3. Il metodo di Fagan può essere complementare al testing del software. (V) 🡪 La fase di verifica include inspection e testing, quindi sono complementari.

**CMM**

1. Il CMM ha l’obiettivo di rivoluzionare il processo di sviluppo del software. (F) 🡪 no, non buttiamo via quanto fatto, bensì si fanno degli evolutionary steps per avere un continuous process improvement. Contiene 5 livelli di maturità che aiutano anche a capire la “capacità” di un software e aiuta l’organizzazione a capire di dare priorità ad impegnarsi nello sviluppo.
2. La presenza del manuale di qualità è una caratteristica del livello 2 del CMM. (F) 🡪 che cazzo è il manuale di qualità
3. Il CMMI è totalmente slegato dal CMM (F) 🡪 Il CMMI è la “guida illustrata” per guidare le aziende e farle salire di livello nella scala CMM, anche se sembrerebbero slegate. CMMI raccoglie le best practices.

**Change management**

1. Nel change management una richiesta di modifica può coinvolgere persone con ruoli diversi. (V) 🡪 Il change management significa che qualcuno chiede, qualcuno dà una priorità, qualcuno registra, mette insieme, assegna a un team di persone una serie di cambiamenti, quindi in effetti si possono coinvolgere persone con ruoli diversi. Coinvolge un valutatore, un product manager (approva o rimanda, raggruppa più richieste in un ordine per gli sviluppatori), gli sviluppatori.
2. Il change management serve per la gestione dei requisiti iniziali del sistema. (F) 🡪 No, le richieste nella fase di change management sono mldifiche successive, non ha nulla a che vedere con i requisiti iniziali

**BPMN**

1. Nel BPMN tutti i gateway (xor, and, or) sono rappresentabili direttamente con reti di Petri. (F) 🡪 Con lo XOR è vero che si può rappresentare perché è una scelta, come se fosse una FC. L’AND è come se fosse un join. L’OR è un po’ più particolare perché un’uscita potrebbe essere con 1-2 token che esce, dipende. Nelle reti di Petri questo non c’è.
2. Nel BPMN il data flow può condizionare l’esecuzione dei task. (V) 🡪 Perché un task possa essere eseguito, non significa che abbia ricevuto un token dal task precedente anche se la maggior parte dei casi è così e non si va oltre, ma occorre andare a vedere che i dati che servono in realtà sono presenti, se no bisogna aspettare.
3. Nel BPMN il control flow e il data flow sono integrati. (?)
4. Nei processi BPN un task composto è equivalente ad una scelta composta. (F)
5. Un modello di processo BPN può combinare i lifecycle di due o più tipi di entità (V)
6. Nei processi BPN tutti i task umani hanno almeno un posto di input e almeno 1 posto di output (F) 🡪 Un’entry task non ha input

Altre:

1. Nella gestione della configurazione è importante minimizzare lo spazio occupato dalle varie versioni dei componenti software. (?)
2. Una classe associativa (association class) ha due relazioni obbligatorie con molteplicità 1. (V)
3. Una relazione recursiva in un modello informativo presuppone sempre che gli oggetti collegati svolgano ruoli reciprocamente diversi, ad esempio se x1 è predecessore di x2, x2 è successore di x1. (F)
4. La disponibilità (availability) e l’affidabilità (reliability) sono attributi di qualità interni.
5. Il concetto di dependability del software comprende i concetti di reliability e security (V)
6. La metrica di Halstead riguarda il numero di linee del codice sorgente di un sistema software. (F) 🡪 Si vanno a contare gli elementi distinti che possono essere operatori o operandi.
7. Baseline sta a versioni di sistema come codeline sta a versioni di componente. (F)
8. Una codeline è la collezione dei componenti software che formano un sistema (F) 🡪 Una codeline è codice, riguarda le versioni di uno stesso componente
9. Il pattern composite contiene una relazione ricorsiva. (V)
10. Un processo singleton non ha nessuna istanza. (F) 🡪 ha una sola istanza
11. Il termine bounded context indica un qualsiasi sottoinsieme di un domain model. (F) 🡪 Un bounded context non è un qualsiasi sottoinsieme di un domain model ma è un sottoinsieme costituito da classi strettamente corelate e con pochi collegamenti con altri bounded context.
12. Un’interazione partecipatica ha lo scopo di collegare il dato trasmesso con un dato trattato dal processo B2B al quale l’interazione si riferisce (T)
13. Check-in e check-out sono operazioni che si eseguono sulle richieste di cambiamento (F) 🡪 Vuol dire leggere/inserire una nuova versione, una versione su cui poi si lavorerà
14. Gli stati gerarchici consentono una decomposizione top-down dei modelli di stato (V)