Elaborarea Orarului (planului de activități)
Obiective:

A înțelege :

- **Necesitatea de utilizare**
- **Impactul asupra calității produsului finit**

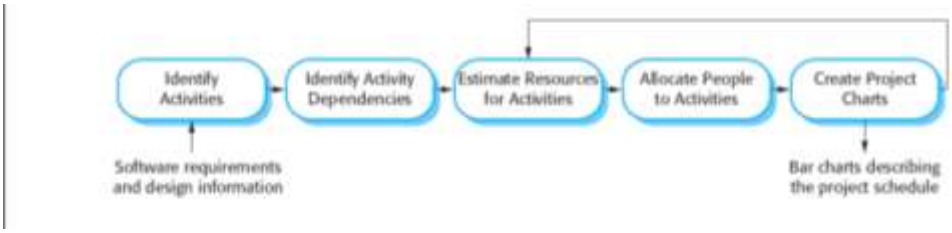
Important de reținut de la început: Atât procesele bazate pe plan strict, cât și **cele agile** au **nevoie de un program inițial** al proiectului, **altfel poate fi numit ORAR**,
Deși **nivelul de detaliu poate fi mai mic într-un plan de proiect agil**.

Cum ar trebui să se înceapă și cum să se procedeze ca la final să se obțină un asemenea document – plan (ORAR)?

În acest context ați putea să îndepliniți următoarele activități:

1. **Definiți-vă obiectivele proiectului.**
   - Notați reperele cheie sau livrabile. Identificați toate părțile interesate.
   - Faceți o listă cu fiecare persoană care trebuie să interacționeze cu echipa de proiect, chiar dacă rolul lor este o formalitate.
2. **Stabiliți-vă termenul limită final**.
   - Decideți când trebuie să terminați complet proiectul.
   - Asigurați-vă că vă acordați suficient timp pentru a da seama de conflictele sau modificările care ar putea apărea mai târziu în timpul gestionării programului.
3. **Enumerați fiecare pas sau sarcină**.
   - Luați acele repere și livrabile pe care le-ați definit la primul pas și
   - împărțiți-le în sarcini și subsarcini mai mici pentru a vă asigura că toate bazele sunt acoperite.
4. **Desemnați un membru al echipei responsabil pentru fiecare sarcină**.
   - Decideți cine va prelua fiecare sarcină și subsarcină și fiți transparent cu termenele limită.
   - Amintiți-vă că membrii echipei probabil au alte proiecte în desfășurare în același timp.
   - Fiți atenți la volumul lor de muncă, astfel încât să nu se simtă supraîncărcați.
5. **Revedeți planul din nou pentru a stabili date limită pentru fiecare sarcină**.
   - Aflați cât timp va dura fiecare sarcină pentru a fi finalizată (data de început și de sfârșit), știind că întârzierile sunt inevitabile.
   - Descompunerea în componente mai mici este de asemenea importantă de luat în considerare, deoarece anumite sarcini vor trebui terminate înainte ca alta să poată începe.
6. **Organizați-vă programul de proiect într-un singur instrument și împărtășiți-l echipei** dvs.
7. **Organizează-ți programul construit** într-un document, într-un anumit format, în care toți cei implicați să poată vedea și să activeze în conformitate cu el

Remarcă. Să fiți conștienți de faptul, că de regulă, un document – ORAR deja finalizat în timpul faza de pornire a proiectului, poate supus mai apoi unor modificări (din diverse motive – cauze) în timpul dezvoltării produsului SOFT
Prezentați situații - exemple când ar putea fi necesară o modificare a orarului
În procesele agile, trebuie să existe un program general care să identifice **când este important ca anumite faze ale proiectului vor fi finalizate**.
O abordare iterativă a programării este **atunci când se cere planificarea pentru fiecare fază**.



O schemă de programare a activităților (ORAR)

Unele restricții:

1. **sarcinile ar trebui să dureze cel puțin o săptămână și nu mai mult de 2 luni**. **Elementele mai fine – sensibile necesită** o perioadă de **timp disproporționată** cheltuită pentru **replanificarea** și **actualizarea** planului de proiect.
2. **Durata maximă** pentru orice sarcină ar trebui să dureze **în jur de 8 până la 10 săptămâni**. Dacă **durează mai mult decât atât, sarcina ar trebui să fie subdivizate** pentru planificarea și programarea proiectului.
3. **Unele** dintre aceste **sarcini sunt** desfășurate **în paralel**, la care **lucrează** diferite **persoane** la diferite componente ale sistemului. Trebuie să coordonezi aceste sarcini paralele și organizați munca astfel încât forța de muncă să fie utilizată optim și să nu introduceți dependențe inutile între sarcini. Este **important să evitați** o situație unde întregul **proiect este amânat pentru că o sarcină critică** este neterminată.
4. **Dacă un proiect este avansat din punct de vedere tehnic**, estimările inițiale vor fi aproape sigur **optimiste chiar și atunci când încerci să iei în considerare toate eventualitățile**. În acest sens, programarea software-ul nu este diferită de programarea oricărui alt tip de proiect mare și avansat.

Avioane noi, poduri și chiar modele noi de mașini întârzie adesea din cauza unor probleme neprevăzute.

Programele, prin urmare, trebuie actualizate continuu ca fiind disponibile, informațiile despre progres. Dacă proiectul care este programat este similar cu un proiect anterior, estimările anterioare pot fi reutilizate.

Cu toate acestea, proiectele pot folosi diferite metode de proiectare și limbaje de implementare, deci experiența anterioară proiectele poate să nu fie aplicabilă în planificarea unui nou proiect.

De reținut:

1. Poate exista posibilitatea ca lucrurile să meargă prost:
   - Oameni se pot îmbolnăvi sau pleca,
   - hardware-ul se poate defecta
   - și software-ul sau hardware-ul de suport esențial poate fi livrat cu întârziere.

2. Pentru proiectele mai **avansate din punct** de vedere tehnic, **anumite părți ale acestuia pot** fi mai **dificile și să dureze mai mult decât se anticipa inițial**.

3. **O regulă de bază bună** este să **estimați ca și cum nimic nu va merge prost**,
apoi să **ridicăm nivelul de estimare pentru a acoperi problemele în mod anticipat**.

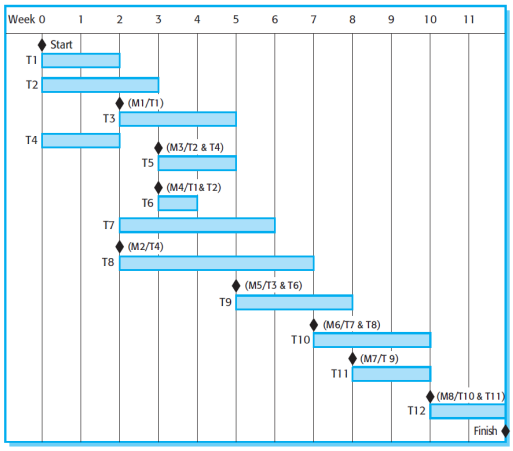
4. **Un alt** factor **de contingență** pentru acoperirea problemele neprevăzute pot fi adăugate la deviz. Acest factor suplimentar de contingență depinde de:
   - **tipul de proiect,**
   - **parametrii procesului (termen limită, standarde etc.) și**
   - **calitate și experiența inginerilor de software care lucrează la proiect**.

Estimările de urgență pot adăuga de la 30% până la 50% la efortul și timpul necesar pentru proiect.

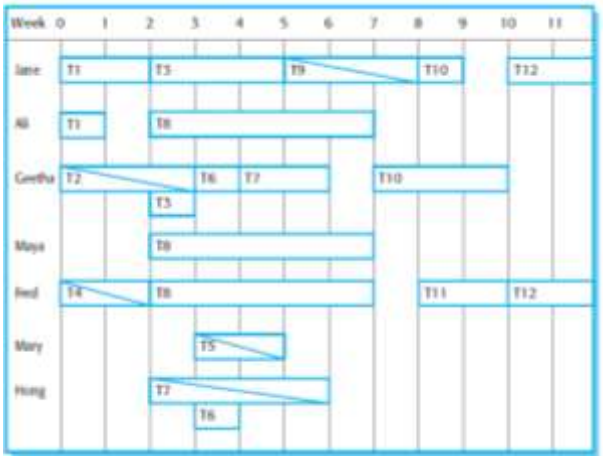
Exemplu de Orar finalizat

Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	15	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T2, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

1. Tabelul primar



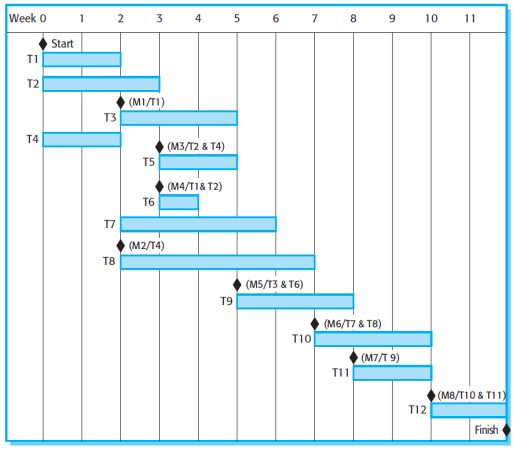
2. Tabelul rezultant din tabelul primar



2. Digrama finală a documentului ORAR

Trei faze la elaborarea documentului final ”ORAR”

Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	15	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)



1. Tabelul inițial(primar) primar
2. Tabelul rezultant din tabelul

Încercăm să revenim cu anumite explicații – comentarii:

(Tabelul din stânga) Informație prezentată într-un tabel sau foaie de calcul care arată:

- sarcini,
- efort,
- durata așteptată și
- dependențe de sarcini

Pentru o mai bună vizualizare (Tabelul inițial - primar)

Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

Comentarii:

Uneori stilul de reprezentare **face dificilă observarea relațiilor** și **dependențelor** între diferitele activități.

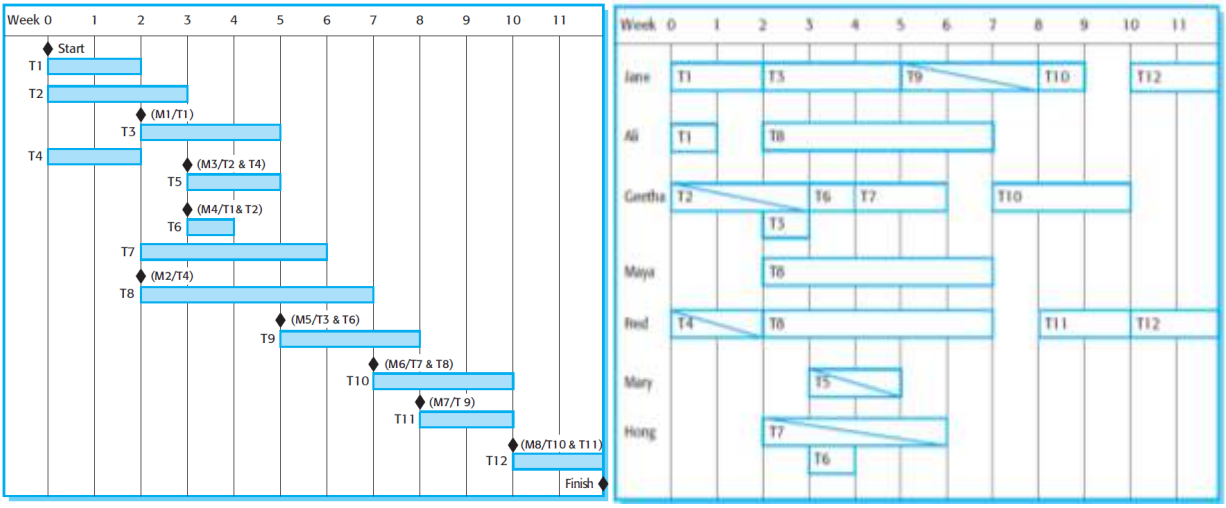
- Jalonul (ținta-marcaj, semn) **M1** este asociat sarcinii **T1**
- și jalonului **M3** este asociat cu o pereche de sarcini, **T2** și **T4**.

În continuare , Prezentăm documentul în format de:

Diagramă cu bare, care se bazează pe **calendar,**

- **arată cine este responsabil** pentru fiecare activitate,
- **timpul scurs așteptat și momentul** în care activitatea este programată să înceapă și să se încheie.

Diagramele cu bare sunt uneori numite „**diagrame Gantt**”, după inventatorul lor, **Henry Gantt**.



Diagrame cu bare rezultante din tabelul de mai sus

De unde se preia asemenea date(despre efort, durată,Dependențe) ?

De obicei, informațiile despre proiect se introduc într-un tabel și apoi se creează o bază de date cu informații despre proiect.
Diagramele cu bare și diagramele de activitate **pot fi apoi generate automat din această bază de date**.

Activitățile de proiect sunt elementul de bază de planificare.

Fiecare activitate are:

1. **O durată în zile calendaristice sau luni.**
2. **O estimare a efortului, care reflectă numărul de persoane-zile sau persoană-lunipentru a finaliza lucrarea.**
3. **Un termen limită până la care activitatea trebuie finalizată.**
4. **Un punct final definit. Acesta reprezintă rezultatul tangibil al finalizării activității.**

Acesta ar putea fi un document, organizarea unei întâlniri de revizuire, succesul executarea tuturor testelor etc.
Atunci când planificați un proiect, ar trebui

- să definiți, etape; adică fiecare etapă în proiect în care se poate face o evaluare a progresului.
- Fiecare piatră de hotar ar trebui să fie documentată printr-un scurt raport care rezumă progresele realizate și munca depusă.

Etapele pot fi asociate cu o singură sarcină sau cu grupuri de activități conexe.
De exemplu,

- Jalonul (ținta-marcaj, semn) **M1** este asociat sarcinii **T1**
- și jalonului **M3** este asociat cu o pereche de sarcini, **T2** și **T4**.

Astfel se obține informația pentru a o prezenta în format de tabel inițial – date primare (a se vedea în următoare pagină)

Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

Un tip special de piatră de hotar este producerea unui livrabil de proiect. Un livrabil este un produs de lucru care este livrat clientului.

Este important un **anumit rezultat semnificativ** în faza proiectului, cum ar fi **specificarea** sau **proiectarea**.

De obicei, **livrabilele care sunt necesare sunt specificate în contractul de proiect și punctul de vedere al clientului** asupra proiectului progresul depinde de aceste livrabile.
Pentru a ilustra modul în care sunt utilizate diagramele cu bare, se prezintă un set ipotetic de sarcini ca prezentat în Figura 23.5.
Acest tabel arată sarcinile, efortul estimat, durata și sarcina interdependente.
În Tabel se poate vedea că **sarcina T3 depinde de sarcină T1**. Prin urmare, sarcina **T1** trebuie finalizată înainte de începerea **T3**.
De exemplu, **T1** ar putea fi pregătirea unui proiect de componentă și **T3**, implementarea proiectului respectiv.

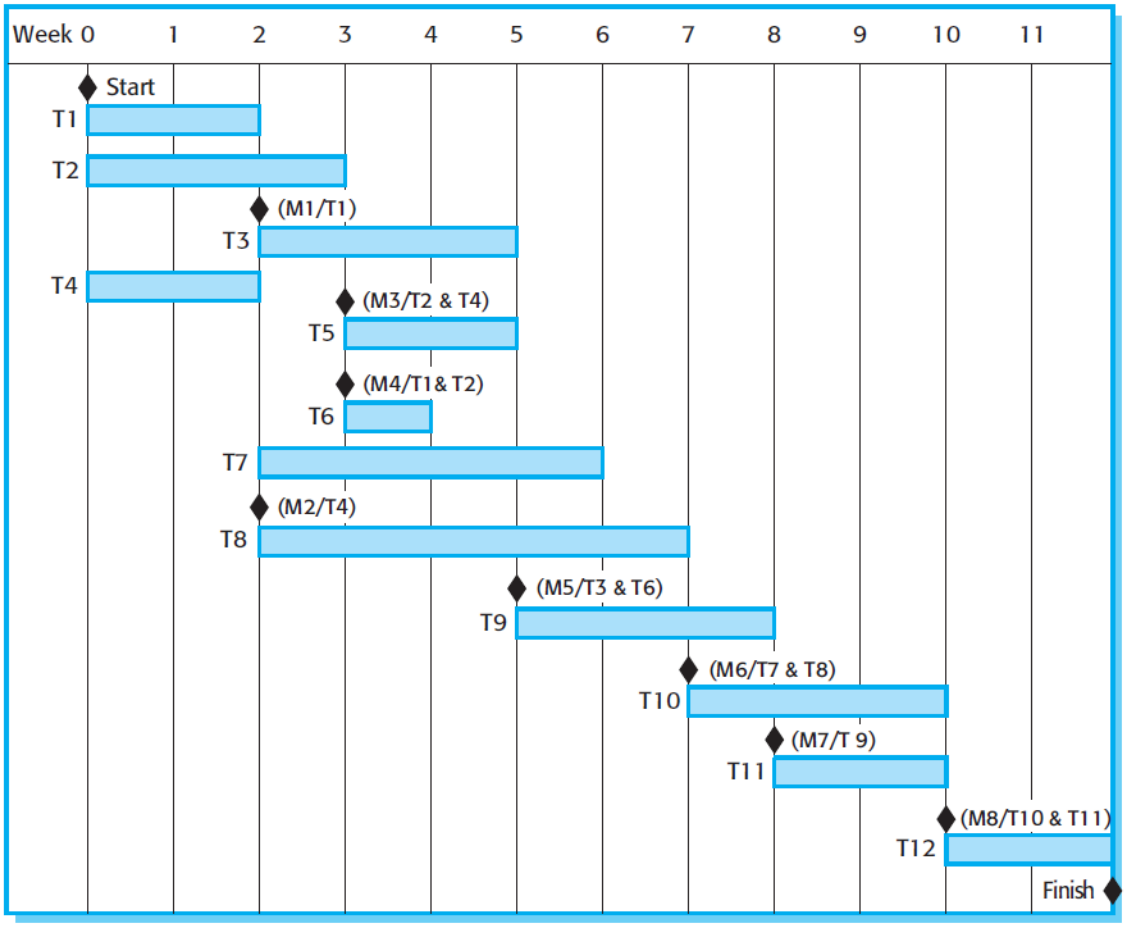
Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

Înainte de începerea implementării, designul ar trebui să fie complet.
Observați că estimarea, duratei
durata pentru unele sarcini este mai mare decât **efortul** necesar **și invers**.

Dacă efortul este mai mic decât durata, asta înseamnă că persoanele alocate acelei sarcini **nu lucrează cu normă întreagă** la el. Dacă **efortul depășește durata**, înseamnă că mai multe membrii echipei lucrează la sarcină în același timp.

Diagramă cu bare arată un calendar al proiectului și începutul și datele de finalizare a sarcinilor. **Citind de la stânga la dreapta**, diagrama cu bare **arată clar când sarcinile începe și se termină**.

Etapele de referință (marcajele)(**M1, M2** etc.) sunt afișate și pe diagrama cu bare.
Observați că sarcinile care sunt independente sunt efectuate în paralel (de exemplu, sarcinile **T1, T2**, și **T4** toate **încep la începutul proiectului**).

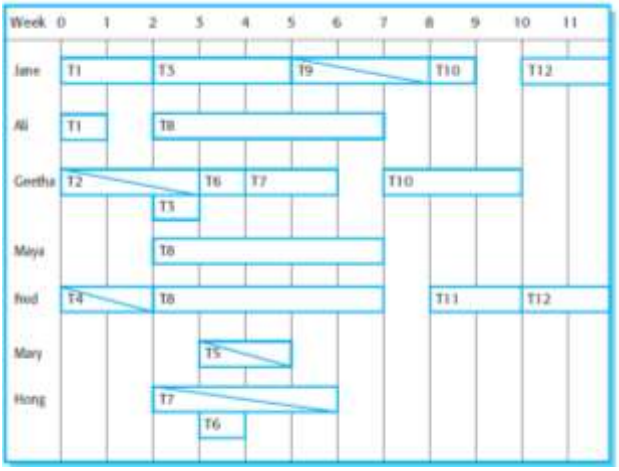


Pe lângă planificarea programului de livrare pentru software, managerii de proiect au să aloce resurse sarcinilor.

Resursa cheie este, desigur, **inginerii software**

cine va face munca și trebuie să fie repartizați activităților proiectului.

Resursa alocarea poate fi, de asemenea, introdusă în instrumentele de management de proiect și generată o diagramă cu bare, care arată când personalul lucrează la proiect (Figura 23.7).



Task	Effort (person-days)	Duration (days)	Dependencies
T1	15	10	
T2	8	15	
T3	20	15	T1 (M1)
T4	5	10	
T5	5	10	T2, T4 (M3)
T6	10	5	T1, T2 (M4)
T7	25	20	T1 (M1)
T8	75	25	T4 (M2)
T9	10	15	T3, T6 (M5)
T10	20	15	T7, T8 (M6)
T11	10	10	T9 (M7)
T12	20	10	T10, T11 (M8)

Distribuirea persoanelor

Oamenii pot fi asociați la mai multe sarcini în același timp și, uneori, nu activează asupra proiectului.

- **Ei pot fi în vacanță,**
- **lucrează la alte proiecte,**
- **participă la cursuri de formare sau**
- **angajarea într-o altă activitate.**

Pot fi sarcini cu jumătate de normă (folosind o linie diagonală care traversează bara).

Organizațiile mari angajează, suplimentar, de obicei un număr de specialiști care lucrează la un proiect **când e nevoie**.

În exemplul prezentat se poate vedea **că Mary** este un specialist, care lucrează **la doar o singură sarcină în proiect**. Acest lucru poate cauza probleme de programare.
Dacă un singur proiect este întârziat în timp ce un specialist lucrează la el, acest lucru poate avea un efect secundar asupra alte proiecte la care se cere si specialistul.
Acestea pot fi apoi amânate deoarece specialistul nu este disponibil.
Dacă o sarcină este întârziată, acest **lucru poate afecta în mod evident sarcinile** ulterioare de care depind aceasta. Nu pot începe până când sarcina întârziată nu este finalizată. Întârzierile pot provoca serioase probleme cu alocarea personalului, mai ales atunci când oamenii lucrează la mai multe proiecte in același timp. Dacă o sarcină **(T)** este întârziată, persoanele alocate pot fi alocate la altă lucrare **(W).** Pentru a finaliza acest lucru poate dura mai mult decât întârzierea, dar o singură dată atribuite, nu pot fi pur și simplu reatribuite înapoi sarcinii inițiale, **T**. Acest lucru poate apoi duce la întârzieri suplimentare în **T** pe măsură ce completează **W**.

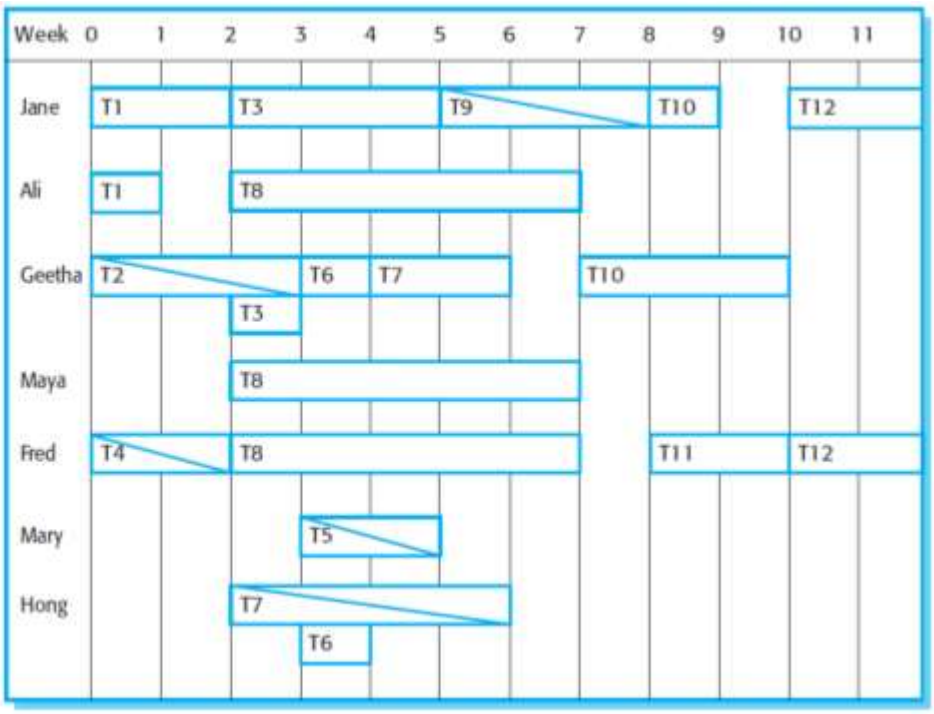


Figure 23.7 Staff allocation chart

23.4 Planificare agilă

Metodele agile de dezvoltare software sunt **abordări iterative** în care software-ul este dezvoltat și **livrat clienților în trepte**. Spre deosebire de planuri abordări, **funcționalitatea** acestor incremente nu este planificată în prealabil, dar **este decis în timpul dezvoltării**.

Decizia cu privire la **ce să includă** într-un increment **depinde de progres și** de **prioritățile clientului**. Argumentul pentru asta abordarea este că **prioritățile și cerințele clientului** se **schimbă**, astfel **încât să aibă sens să aibă un plan flexibil** care să poată face față acestor schimbări. **Cartea lui Cohn** {Cohn, 2005 ##1735} este o discuție cuprinzătoare despre problemele de planificare în **proiecte agile**.

Cele **mai frecvent** utilizate abordări **agile**, **cum ar fi Scrum** (Schwaber, 2004) și **programarea extremă** (Beck, 2000) au o **abordare** în **două etape** a planificării, corespunzătoare fazei de pornire a dezvoltării și dezvoltării bazate pe plan planificare:

1. **Planificarea lansării**, care **privește** în viitor **timp de câteva luni** și decide asupra caracteristicilor care ar trebui să fie incluse într-**o versiune** a **unui sistem**.
2. **Planificarea iterației**, care are o **perspectivă pe termen mai scurt** și se concentrează pe planificare următorul **increment al unui sistem**. Aceasta este de obicei **2 până la 4 săptămâni** de muncă pentru echipă.

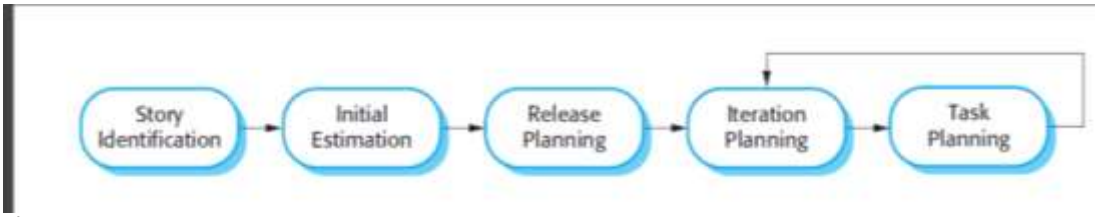


Figure 23.8
Planning in XP

Referitor la programare extremă (XP). Aceasta se numește „joc planificare”.
și de obicei implică întreaga echipă de dezvoltare, inclusiv reprezentanții clienților. Figura 23.8 prezintă etapele jocului de planificare.

Specificațiile de sistem din XP se bazează pe intențiile (poveștile) utilizatorilor care reflectă caracteristicile funcționalităților care ar trebui incluse în sistem. La începutul proiectului, echipa și clientul încearcă să identifice un set de intenții, care acoperă toate funcționalitățile care vor să fie incluse în sistemul final. Unele funcționalități vor lipsi inevitabil, dar acest lucru nu este important în această etapă.

Următoarea etapă este o etapă de estimare. Echipa de proiect citește și discută despre poveștile și le clasifică în ordinea timpului pe care cred că le va lua pentru implementarea poveștii. Acest lucru poate implica împărțirea poveștilor mari în povești mai mici. Relativ estimarea este adesea mai ușoară decât estimarea absolută. Oamenilor le este adesea greu estimați cât de mult efort sau timp este necesar pentru a face ceva. Cu toate acestea, atunci când sunt prezentați cu mai multe lucruri de făcut, ei pot emite judecăți despre care povești vor fi ia cel mai mult timp și cel mai mult efort. Odată ce clasamentul a fost finalizat, echipa apoi alocă povestirilor puncte de efort noțional. O poveste complexă poate avea 8 puncte si o poveste simpla 2 puncte. Faceți acest lucru pentru toate poveștile din lista clasată.

Odată estimate poveștile, efortul relativ este tradus în prima estimarea a efortului total necesar prin utilizarea noțiunii de „viteză". În XP, viteza este numărul de puncte de efort implementate de echipă, pe zi. Acest lucru poate fi estimat fie din experiența anterioară, fie prin dezvoltarea uneia sau două povești de văzut cât timp este necesar. Estimarea vitezei este aproximativă, dar este rafinată în timpul procesul de dezvoltare. Odată ce aveți o estimare a vitezei, puteți calcula efortul total în zile-persoană pentru implementarea sistemului.

Planificarea lansării implică selectarea și rafinarea poveștilor care vor reflecta caracteristici care urmează să fie implementate într-o versiune a unui sistem și ordinea în care poveștile ar trebui implementat. Clientul trebuie să fie implicat în acest proces. O eliberare este apoi aleasă data și poveștile sunt examinate pentru a vedea dacă estimarea efortului este consecventă cu acea dată.
Dacă nu, poveștile sunt adăugate sau eliminate din listă.

Planificarea iterativă este prima etapă a procesului de dezvoltare a iterației. Poveștile sunt alese pentru a fi implementate pentru acea iterație, cu numărul de povești reflectând timpul de livrare a unei iterații (de obicei 2 sau 3 săptămâni) și viteza echipei. Când data de livrare a iterației este atinsă, acea iterație este completă, chiar dacă toate poveștile nu au fost implementate. Echipa ia în considerare poveștile care au fost implementate și adună punctele lor de efort. Viteza poate fi apoi recalculată și aceasta este utilizată în planificarea următoarei ediții a sistemului.

La începutul fiecărei iterații, există o etapă de planificare mai detaliată în care dezvoltatorii descompun poveștile în sarcini de dezvoltare. O sarcină de dezvoltare ar trebui durează 4-16 ore. Sunt enumerate toate sarcinile care trebuie îndeplinite pentru a implementa toate poveștile din acea iterație. Dezvoltatorii individuali se înscriu apoi pentru specificul sarcinile pe care le vor implementa. Fiecare dezvoltator își cunoaște viteza individuală nu ar trebui să se înscrie pentru mai multe sarcini decât pot implementa în timp.

Există două beneficii importante din această abordare a alocării sarcinilor:

1. Întreaga echipă primește o imagine de ansamblu asupra sarcinilor care trebuie îndeplinite într-o iterație. Prin urmare, au o înțelegere a ceea ce fac alți membri ai echipei și cu cine să discutați dacă sunt identificate dependențe de sarcini.
2. Dezvoltatorii individuali aleg sarcinile de implementat; nu sunt doar sarcini alocate de către un manager de proiect. Prin urmare, au un sentiment de proprietate în aceste sarcini și acest lucru este de natură să-i motiveze să ducă la bun sfârșit sarcina.

La jumătatea unei iterații, progresul este revizuit. În acest stadiu, jumătate din poveste punctele de efort ar fi trebuit finalizate. Deci, dacă o iterație implică 24 de puncte de poveste și Ar fi trebuit îndeplinite 36 de sarcini, 12 puncte de poveste și 18 sarcini. Dacă nu este cazul,apoi clientul trebuie consultat și unele povești eliminate din iterație.

Această abordare a planificării are avantajul că software-ul este întotdeauna lansat conform planului și nu există derapaje de program. Dacă lucrarea nu poate fi finalizată în timpul permis, filozofia XP este de a reduce sfera lucrării, mai degrabă decât extinde programul. Cu toate acestea, în unele cazuri, creșterea poate să nu fie suficientă pentru a fi util. Reducerea domeniului de aplicare poate crea muncă suplimentară pentru clienți, dacă trebuie să utilizeze un sistem incomplet sau își schimbă practicile de lucru între o versiune a sistemului și alta.

O **dificultate majoră** în planificarea agilă este că se bazează **pe implicarea clienților presupunând și** disponibilitate. În practică, acest lucru poate fi dificil de aranjat, deoarece reprezentantul clientului trebuie uneori să acorde prioritate altor lucrări. **Clienții pot fi mai familiari cu planuri de proiect tradiționale și poate fi dificil să se angajeze într-o planificare agilă proiect**.

Planificarea agilă funcționează bine cu echipe de dezvoltare mici, stabile, care pot obține împreună și discutate istoriile care urmează să fie implementate. Totuși, acolo **unde echipele sunt mari** și/sau **distribuit geografic** sau atunci **când membrii echipei se schimbă frecvent**, acesta **este practic imposibil ca toată lumea să fie implicată în planificarea prin colaborare** care **este esențial** pentru managementul **agil** al proiectelor. În consecință, **proiectele mari sunt de obicei planificate folosind abordări tradiționale** ale managementului de proiect.

Programare pereche

O altă practică inovatoare care a fost introdusă în XP este că programatorii lucrați în perechi pentru a dezvolta software-ul. Ei stau de fapt împreună la aceeași stație de lucru pentru a dezvolta software-ul. Totuși, aceleași perechi nu se programează întotdeauna împreună. Mai degrabă, perechile sunt create dinamic, astfel încât toți membrii echipei să lucreze cu reciproc în timpul procesului de dezvoltare.

Utilizarea programării perechilor are o serie de avantaje:

1. Susține ideea de proprietate colectivă și responsabilitate pentru sistem.
Aceasta reflectă ideea lui Weinberg (1971) de programare fără ego, în care software-ul este deținut de echipă în ansamblu și indivizii nu sunt responsabili.
pentru probleme cu codul. În schimb, echipa are responsabilitatea colectivă pentru rezolvarea acestor probleme.
2. Acționează ca un proces informal de revizuire deoarece fiecare linie de cod este privită de către cel putin doua persoane. Inspecțiile și revizuirile codului (acoperite în capitolul 24) sunt foarte cu succes în descoperirea unui procent mare de erori software. Cu toate acestea, ei organizarea necesită timp și, de obicei, introduc întârzieri în procesul de dezvoltare. Deși programarea în pereche este un proces mai puțin formal, care probabil nu găsește atât de multe erori ca inspecțiile de cod, este mult mai ieftin.
procesul de inspecție decât inspecțiile oficiale ale programului.
3. Ajută la sprijinirea refactorizării, care este un proces de îmbunătățire a software-ului. Dificultatea de a implementa acest lucru într-un mediu de dezvoltare normal este acel efort în refactorizarea este cheltuită pentru beneficii pe termen lung. Un individ care practică refactorarea poate fi considerat a fi mai puțin eficient decât unul care pur și simplu continuă să se dezvolte cod. Acolo unde se utilizează programarea în pereche și proprietatea colectivă, alții beneficiază imediat de la refactorizare, astfel încât acestea sunt susceptibile de a sprijini procesul.
Ai putea crede că programarea în pereche ar fi mai puțin eficientă decât programarea individuală. Într-un timp dat, o pereche de dezvoltatori ar produce jumătate din cât de mult cod doi indivizi care lucrează singuri. Au existat diverse studii privind productivitatea programatori plătiți cu rezultate mixte. Folosind studenți voluntari, Williams și ea colaboratorii (Cockburn și Williams, 2001; Williams și colab., 2000) au descoperit că productivitatea cu programarea în pereche pare să fie comparabilă cu cea a doi oameni.
lucrand independent.
Motivele sugerate sunt că perechile discută despre software înainte de dezvoltare, astfel încât probabil să aveți mai puține porniri false și mai puține reluări. În plus, numărul de erori evitate prin inspecția informală este astfel încât se petrece mai puțin timp repararea erorilor descoperite în timpul procesului de testare.
Cu toate acestea, studii cu programatori mai experimentați (Arisholm et al., 2007;
Parrish et al., 2004) nu au replicat aceste rezultate. Ei au descoperit că a existat o pierdere semnificativă a productivității în comparație cu doi programatori care lucrează singuri. Au fost unele beneficii de calitate, dar acestea nu au compensat pe deplin programarea perechilor deasupra capului. Cu toate acestea, împărtășirea cunoștințelor care are loc în timpul programării perechilor este foarte importantă, deoarece reduce riscurile generale pentru un proiect atunci când membrii echipei părăsi. În sine, acest lucru poate face ca programarea perechilor să merite.

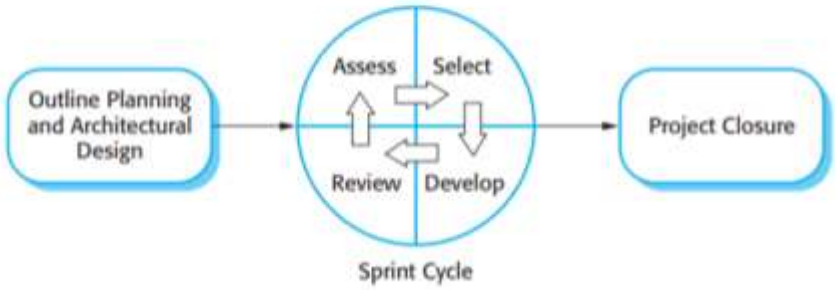


Figure 3.8 The Scrum process

Aceasta este urmată de o serie de cicluri de sprint, în care fiecare ciclu dezvoltă un increment a sistemului. În cele din urmă, faza de închidere a proiectului încheie proiectul, se finalizează documentația necesară, cum ar fi cadrele de ajutor ale sistemului și manualele de utilizare și evaluările lecțiile învățate din proiect.

Caracteristica inovatoare a Scrum este faza sa centrală, și anume ciclurile de sprint.
Un Scrum sprint este o unitate de planificare în care se evaluează munca de făcut, caracteristici sunt selectate pentru dezvoltare, iar software-ul este implementat. La sfârșitul a sprint, funcționalitatea finalizată este livrată părților interesate.
Caracteristici cheie ale acestui proces sunt următoarele:
1. Sprinturile au o lungime fixă, de obicei 2-4 săptămâni. Acestea corespund dezvoltării unei versiuni a sistemului în XP.
2. Punctul de plecare pentru planificare este stocul de produse, care este lista de lucru de realizat pe proiect. În timpul fazei de evaluare a sprintului, aceasta este revizuite, iar prioritățile și riscurile sunt atribuite. Clientul este strâns implicat în acest proces și poate introduce noi cerințe sau sarcini la începutul fiecare sprint.
3. Faza de selecție implică toată echipa de proiect care lucrează cu clientul pentru a selecta caracteristicile și funcționalitățile care urmează să fie dezvoltate în timpul sprintului.
4. Odată ce acestea sunt convenite, echipa se organizează pentru a dezvolta software-ul.
Scurte întâlniri zilnice care implică toți membrii echipei au loc pentru a analiza progresul și, dacă este necesar, reprioritizează munca. În această etapă echipa este izolată de clientul și organizația, cu toate comunicările canalizate prin așa-numitul „Scrum master”. Rolul Scrum Master este de a proteja echipa de dezvoltare de la distragerile externe. Modul în care este lucrarea realizat depinde de problemă și de echipă. Spre deosebire de XP, Scrum nu face sugestii specifice despre cum să scrieți cerințele, dezvoltarea mai întâi de testare etc.
Cu toate acestea, aceste practici XP pot fi folosite dacă echipa consideră că sunt adecvate.
5. La sfârșitul sprintului, munca depusă este revizuită și prezentată părților interesate.
Următorul ciclu de sprint începe apoi.

Ideea din spatele Scrum este că întreaga echipă ar trebui să fie împuternicită să facă decizii, astfel încât termenul „manager de proiect” a fost evitat în mod deliberat. Mai degrabă, cel „Scrum master” este un facilitator care organizează întâlniri zilnice, urmărește restanța munca de făcut, înregistrează deciziile, măsoară progresul în raport cu restanța și comunică cu clienții și managementul din afara echipei.
Întreaga echipă participă la întâlnirile zilnice, care uneori sunt „în picioare” întâlniri pentru a le menține scurte și concentrate. În timpul întâlnirii, toți membrii echipei

împărtășesc informații, descrie progresul lor de la ultima întâlnire, probleme care au
apărut și ce este planificat pentru ziua următoare. Aceasta înseamnă că toată lumea de pe
echipa știe ce se întâmplă și, dacă apar probleme, poate replanifica munca pe termen scurt
face față lor. Toată lumea participă la această planificare pe termen scurt – nu există o direcție de sus în jos din partea Scrum Master.
Există multe rapoarte anecdotice despre utilizarea cu succes a Scrum disponibile pe
Web. Rising și Janoff (2000) discută despre utilizarea sa cu succes în telecomunicații
mediu de dezvoltare software și prezintă avantajele acestuia după cum urmează:
1. Produsul este împărțit într-un set de gestionabile și de înțeles
bucăți.
2. Cerințele instabile nu țin progresul.
3. Întreaga echipă are vizibilitate asupra tuturor lucrurilor și, în consecință, comunicarea echipei este îmbunătățită.
4. Clienții văd livrarea la timp a creșterilor și obțin feedback despre modul în care
produsul funcționează.
5. Încrederea între clienți și dezvoltatori este stabilită și este o cultură pozitivă
creat în care toată lumea se așteaptă ca proiectul să reușească.
Scrum, așa cum a fost proiectat inițial, a fost conceput pentru a fi utilizat cu echipe amplasate în comun, unde
toți membrii echipei se puteau reuni în fiecare zi în întâlniri stand-up. In orice caz,
multă dezvoltare de software implică acum echipe distribuite cu membri ai echipei
situate în diferite locuri din lume. În consecință, sunt în curs de desfășurare diverse experimente pentru a dezvolta Scrum pentru medii de dezvoltare distribuite (Smits
și Pshigoda, 2007; Sutherland et al., 2007).

Metoda Scrum este o metodă agilă care oferă un cadru de management de proiect. Este
centrat în jurul unui set de sprinturi, care sunt perioade de timp fixe când este o creștere a sistemului dezvoltat. Planificarea se bazează pe prioritizarea unui acumulat de muncă și pe selectarea sarcinilor cu cea mai mare prioritate pentru un sprint.

Scrum, așa cum a fost proiectat inițial, a fost conceput pentru a fi utilizat cu echipe amplasate în comun, unde
toți membrii echipei se puteau reuni în fiecare zi în întâlniri stand-up. In orice caz,
multă dezvoltare de software implică acum echipe distribuite cu membri ai echipei
situate în diferite locuri din lume. În consecință, sunt în curs de desfășurare diverse experimente pentru a dezvolta Scrum pentru medii de dezvoltare distribuite (Smits
și Pshigoda, 2007; Sutherland și colab., 2007)