Universitatea din București

# Metode de dezvoltare software

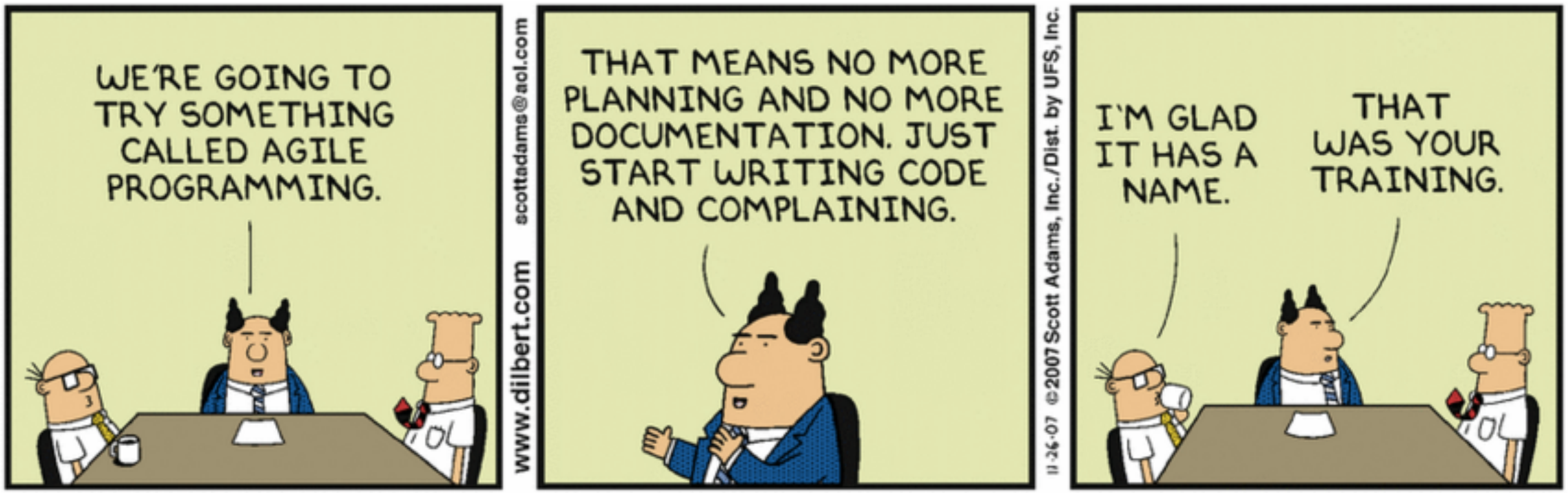
## Metodologii agile

14.03.2023

Alin Ștefănescu


Prezentare bazată pe “Ian Sommerville: Software Engineering 10th edition”

# Procese agile… în practică




de la dilbert.com

# Metodologii “agile”

- scopul metodelor agile este de a reduce cheltuielile în procesul de dezvoltare a software-ului (de exemplu, prin limitarea documentației) și de a răspune rapid cerințelor în schimbare.

- aceste metode:
  - se concentrează mai mult pe cod decât pe proiectare
  - se bazează pe o abordare iterativă de dezvoltare de software
  - produc rapid versiuni care funcționează, acestea evoluând repede pentru a satisface cerințe în schimbare.

- în 2001, este publicat și semnat de mai mulți practicieni “Manifestul Agil”, care exprimă succint principiile metodelor agile.

# Agile Manifesto

**http://agilemanifesto.org/iso/ro/**

"*Noi scoatem la iveală modalități mai bune de dezvoltare de software prin experiență proprie și ajutându-i pe ceilalți.*

*Prin această activitate am ajuns să apreciem:*

- **indivizii şi interacţiunea** înaintea proceselor şi uneltelor
- **software-ul funcţional** înaintea documentaţiei vaste
- **colaborarea cu clientul** înaintea negocierii contractuale
- **receptivitatea la schimbare** înaintea urmăririi unui plan

*Cu alte cuvinte, deşi există valoare în elementele din dreapta, le apreciem mai mult pe cele din stânga."*

# Cele 12 principii ale manifestului agil

**http://agilemanifesto.org/iso/ro/principles.html**

1. Prioritatea noastră este satisfacţia clientului prin livrarea rapidă şi continuă de software valoros.
2. Schimbarea cerinţelor este binevenită chiar şi într-o fază avansată a dezvoltării. Procesele agile valorifică schimbarea în avantajul competitiv al clientului.
3. Livrarea de software funcţional se face frecvent, de preferinţă la intervale de timp cât mai mici, de la câteva săptămâni la câteva luni.
4. Clienții şi dezvoltatorii trebuie să colaboreze zilnic pe parcursul proiectului.

# Cele 12 principii ale manifestului agil

**http://agilemanifesto.org/iso/ro/principles.html**

5. Construieşte proiecte în jurul oamenilor motivaţi. Oferă-le mediul propice şi suportul necesar şi ai încredere că obiectivele vor fi atinse.

6. Cea mai eficientă metodă de a transmite informaţii înspre şi în interiorul echipei de dezvoltare este comunicarea directă, faţă în faţă.

7. Software funcţional este principala măsură a progresului.

8. Procesele agile promovează dezvoltarea durabilă. Sponsorii, dezvoltatorii şi utilizatorii trebuie să poată menţine un ritm constant pe termen nedefinit.

# Cele 12 principii ale manifestului agil

**http://agilemanifesto.org/iso/ro/principles.html**

9. Atenţia continuă pentru excelenţă tehnică şi design bun îmbunătăţeşte agilitatea.
10. Simplitatea — arta de a maximiza cantitatea de muncă nerealizată — este esenţială.
11. Cele mai bune arhitecturi, cerinţe şi design se obțin de către echipe care se auto-organizează.
12. La intervale regulate, echipa reflectează la cum să devină mai eficientă, apoi îşi adaptează şi ajustează comportamentul.

# Aplicabilitatea metodelor agile

- în companiile care dezvoltă **produse software de dimensiuni mici sau mijlocii**.
- în cadrul companiilor unde se dezvoltă **software pentru uz intern (proprietary software)**, deoarece există un angajament clar din partea clientului (intern) de a se implica în procesul de dezvoltare și deoarece nu există o mulțime de reguli și reglementări externe care afectează software-ul.
- datorită orientării lor pe echipe mici și bine integrate, există probleme în scalarea metodelor agile pentru sistemele mari (deși și în acest caz se pot realiza anumite componente ale sistemului în mod agil).

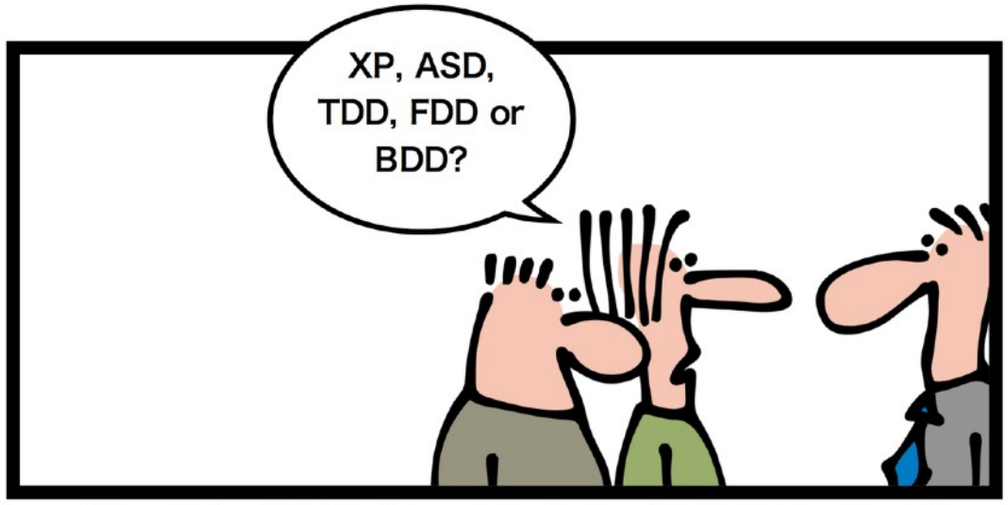
# Posibile probleme cu metodologiile agile

- dificultatea de a păstra interesul clienților implicați în acest procesul de dezvoltare pentru perioade lungi
- membrii echipei nu sunt întotdeauna potriviți pentru implicarea intensă care caracterizează metodele agile
- prioritizarea modificărilor poate fi dificilă atunci când există mai multe părți interesate
- menținerea simplității nu e ceva simplu :-)
- contractele pot fi o problemă ca și în alte metode de dezvoltare incrementală

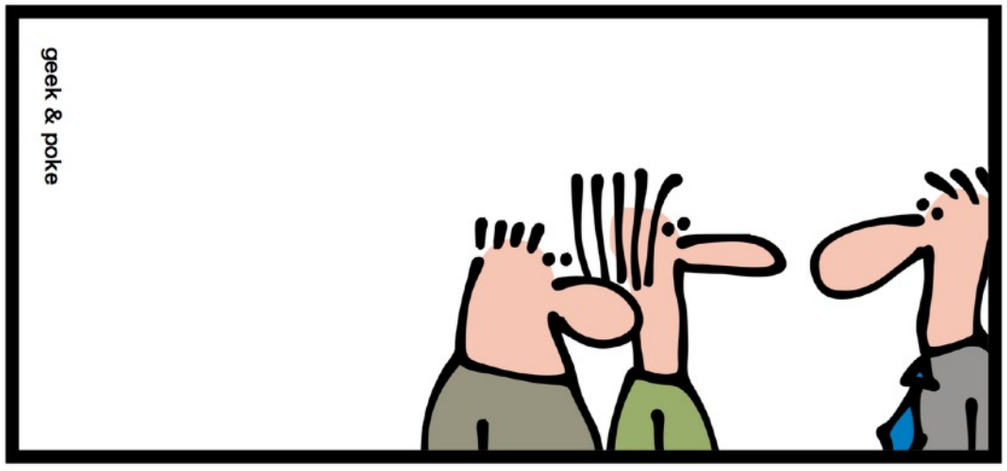
# Metodologii “agile”

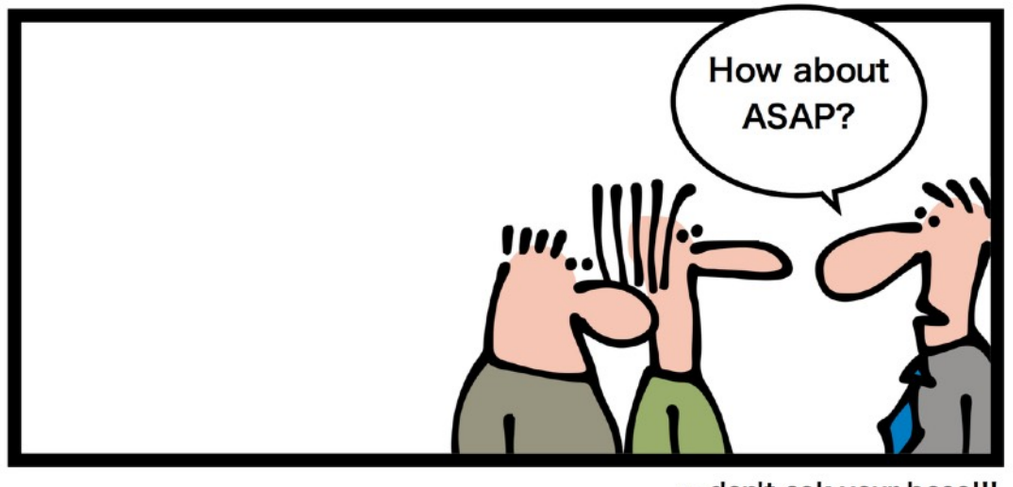
Exemple de **metodologii agile**:

- **Extreme Programming (XP)** - 1996
- Adaptive Software Development (ASD)
- Test-Driven Development (TDD)
- Feature Driven Development (FDD)
- Behavior Driven Developement (BDD)
- Crystal Clear
- **Scrum - 1995**
- etc.



When you're thinking about new software development approaches...



··· don't ask your boss!!!

# Programare extremă

de la spottedpaint.com

# Programare extrema (XP)

- poate cea mai cunoscută metodă agilă.
- **Extreme Programming (XP)** are o abordare "extremă" de dezvoltare incrementală:
  - noile versiuni pot fi construite de mai multe ori pe zi;
  - acestea sunt livrate clienților la fiecare 2 săptămâni;
  - toate testele trebuie să fie executate pentru fiecare versiune și o versiune e livrabilă doar în cazul în care testele au rulat cu succes.

Informații: `http://www.extremeprogramming.org`

# XP și principiile “agile”

- “dezvoltarea incrementală” este susținută prin intermediul livrării de software în mod frecvent cu mici incremente.
- “implicarea clientului” înseamnă angajamentul “full-time” al clientului cu echipa de dezvoltare.
- “oameni, nu procese” prin programare în doi (pereche de programatori), proprietatea colectivă și un proces care să evite orele lungi de lucru.
- “receptivitate la schimbare” prin livrări frecvente.
- “menținerea simplității” prin refactoring constant de cod.

# Practici XP

- procesul de planificare (The Planning Game)
- client disponibil pe tot parcursul proiectului (On-Site Customer)
- implementare treptată (Small Releases)
- limbaj comun (Metaphor)
- integrare continuă (Continuous Integration)
- proiectare simplă (Simple Design)
- testare (Testing)
- rescriere de cod pentru îmbunătățire (Refactoring)
- programare în pereche (Pair Programming)
- drepturi colective (Collective Ownership)
- 40 ore/săptămână (40-Hour Week)
- standarde de scriere a codului (Coding Standard)

# Planificare


- “Joc” de planificare a livrărilor: clienți și dezvoltatori.
- “Joc” de planificare a iterațiilor: doar dezvoltatorii
- Clientul înțelege domeniul de aplicare, prioritățile, nevoile business ale versiunilor care trebuie livrate:
  - sortează “cartonașele” cu sarcini după priorități
- Dezvoltatorii estimează riscurile și eforturile:
  - sortează “cartonașele” după risc
  - dacă o sarcină ia mai mult de 2-4 săptămâni, e distribuită pe mai multe “cartonașe”

# Livrări frecvente

- livrări cât de des este posibil, cu condiția să existe o plus-valoare pentru client
- acest lucru asigură feedback-ul rapid
- clientul are funcționalitatea esențială cât mai curând posibil
- timp între livrări de la o săptămână până la o lună
  (proiectele non-XP au termene de jumătate de an sau mai mult)

# Metafora

- “metafora” este de fapt cuvântul-cheie XP pentru ceea ce alți ingineri numesc “arhitectura sistemului”
- se evită cuvântul “arhitectură” pentru a sublinia faptul că nu avem de-a face doar cu structura generală a sistemului
- “metafora” sugerează o coerență generală, ușor de comunicat, plus mai multă maleabilitate

# Integrare continuă

- codul este integrat și testat în cel mult câteva ore sau o zi de când este scris.
- de exemplu, atunci când dezvoltatorii au terminat o parte din implementare:
  - o integrează cu codul existent
  - rulează teste și corectează eventualele probleme
  - dacă toate testele sunt pozitive, adaugă modificările în sistemul care se ocupă cu managementul codului sursă.

# Proiectare simplă

- principiul de bază: “proiectează cel mai simplu lucru care funcționează acum. Nu proiecta și pentru mâine, pentru că s-ar putea să nu fie nevoie”
- deci, proiectează doar pentru a satisface cerințele și nimic în plus

# Testare

- se testează tot ceea ce ar putea fi problematic.
- programatorii scriu **teste unitare** folosind un cadru de testare automatizată (de exemplu, JUnit) pentru a minimiza efortul de scriere și verificare a testelor.
- clienții, cu ajutorul dezvoltatorilor, scriu **teste funcționale**.
- de multe ori, se aplică metoda "**test-driven development**":
  - se scriu teste înaintea codului pentru a clarifica cerințele.
  - testele sunt scrise ca programe în loc de date, astfel încât acestea să poată fi executate automat.
  - fiecare test include o condiție de corectitudine.
  - toate testele anterioare și cele noi sunt rulate automat atunci când sunt adăugate noi funcționalități, verificând astfel că noua funcționalitate nu a introdus erori.

# Îmbunătățirea codului

- îmbunătățirea codului prin “refactoring” este foarte importantă deoarece XP recomandă începerea implementarii foarte repede.
- simplificarea codului duce la o mai bună înțelegere a lui, astfel compensând lipsa documentației în anumite situații.
- exemplu (“*three strikes and you refactor*”) eliminarea duplicării: daca o bucată de cod similară apare în trei locuri, se scrie codul respectiv într-o metodă care e folosită în cele trei locuri.

# Programarea în echipe de doi

- tot codul este scris de două persoane folosind un singur calculator
- sunt **două roluri** în această echipă:
  - unul scrie cod și
  - celălalt îl ajută gândindu-se la diverse posibilități de îmbunătățire:
    - merge abordarea curentă?
    - care sunt testele care s-ar putea să nu funcționeze?
    - există simplificări posibile?

# Avantajele programării “în doi”

- susține ideea de proprietate și responsabilitate în echipă pentru sistemul colectiv
- proces de revizuire îmbunătățit, deoarece fiecare linie de cod este privită de cel puțin două persoane (“*four eyes principle*”)
- ajută la îmbunătățirea codului
- transfer de cunoștințe și training implicit (important când membrii echipei se schimbă)
- “more fun”?

# Pair programming… în practică


PetraCross.com



CommitStrip

# Drepturi colective asupra codului

- nu există situația în care cineva are anumite “module” pe care nimeni altcineva nu le poate atinge.
- dacă cineva vede o modalitate de a îmbunătăți ceva, va face toate modificările necesare în sistem (desigur e nevoie de un sistem bun de versionare și management al codului).

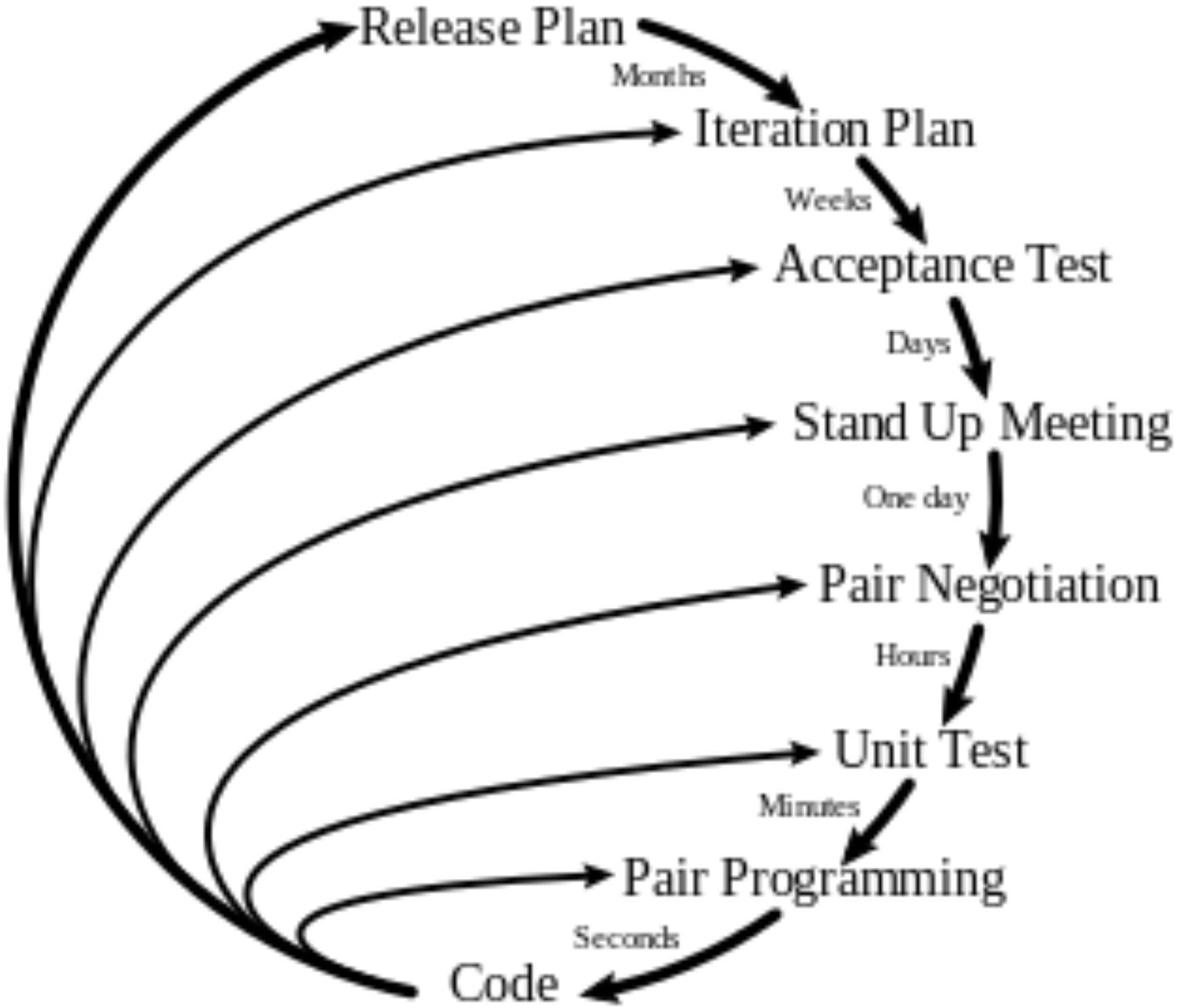
# 40 de ore pe săptamână

- săptămâna de lucru de maxim 40 de ore
- oamenii au nevoie de odihna necesară pentru a munci eficient și a produce cod de calitate înaltă
- există săptămâni când e necesar să se lucreze mai mult, dar aceasta trebuie să fie o excepție

# Codul scris în mod standardizat

- întreaga echipă aderă la un set unic de convenții cu privire la modul în care codul este scris
- scopul este de facilita programarea “în doi” și proprietatea colectivă a codului

# Planificare și iterații XP



# Avantaje XP

- soluţie bună pentru proiecte mici
- programare organizată
- reducerea numărului de greşeli
- clientul are control (de fapt, toată lumea are control, pentru că toţi sunt implicaţi în mod direct)
- dispoziție la schimbare chiar în cursul dezvoltării

# Dezavantaje XP

- nu este scalabilă
- necesită mai multe resurse umane ”pe linie de cod” (datorită d.ex. programării în doi)
- implicarea clientului în dezvoltare (costuri suplimentare și schimbări prea multe)
- lipsa documentelor “oficiale”
- necesită experienţă în domeniu (“senior level” developers)
- poate deveni uneori o metodă ineficientă (rescriere masivă de cod)

# Tool-uri pentru metode “agile” și Scrum

Există foarte multe tool-uri pentru metodele și practicile “agile”.

Câteva exemple:

`http://pivotaltracker.com`

`http://agilescout.com/best-agile-scrum-tools/`

`https://www.scrumwise.com/scrum/#`

`http://www.acunote.com/how-it-works`

etc.

# Câteva exemple

Tehnicile “agile” sunt folosite în foarte multe companii, deși multe dintre ele nu folosesc explicit termenul de “agile”.

Câteva referințe:

- Google: `https://abseil.io/resources/swe-book & https://arxiv.org/pdf/1702.01715`
- Facebook: `https://engineering.fb.com/web/rapid-release-at-massive-scale/`
- Microsoft: `https://learn.microsoft.com/en-us/devops/plan/scaling-agile`
- Amazon: `http://highscalability.com/blog/2019/3/4/how-is-software-developed-at-amazon.html`
- SAP: `http://scn.sap.com/community/agile-software-engineering`

# O comparație de final

| Metode agile | Metode cascadă | Metode formale |
|---|---|---|
| criticalitate scăzută | criticalitate ridicată | criticalitate extremă |
| dezvoltatori seniori | dezvoltatori juniori | dezvoltatori seniori |
| cerințe în schimbare | cerințe relativ fixe | cerințe limitate |
| echipe mici | echipe mari | echipe mici |
| cultură orientată spre schimbare | cultură orientată spre ordine | cultură orientată spre calitate și precizie |

# Post Scriptum

De la `https://www.rainerhahnekamp.com/en/modern-software-development/`

*“Modern software development is about knowing the landscape of tools and libraries as well as using them. Knowing how to create an optimal algorithm is not mandatory anymore.*

*Programming has transitioned from the traditional coding of the 90s which required sophisticated application of the right data structures and algorithms to today’s orchestration of libraries, frameworks. Programming talent is still essential for reading and understanding alien code, but no longer enough to be an effective developer.”*