Programmatūras attīstības tehnoloģijas. Spēja programmatūras izstrāde. XP

Dr.sc.ing., asoc. prof. Oksana Ņikiforova
DITF LDI
Lietišķo datorzinātņu katedra
Rīga - LV1048, Meža 1/3, 510.kab., tel.67 08 95 98
oksana.nikiforova@rtu.lv

Programmatūras izstrādes metodoloģiju attīstība

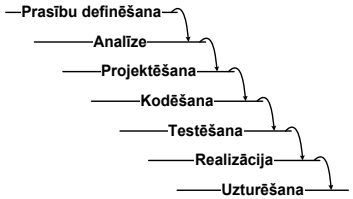
- Bez metodoloģijas – "code and fix" princips, kas ir īslaicīgo risinājumu kopums un noved pie liela kļūdu skaita gatavā produktā
- Monumentālās (smagas) metodoloģijas programmatūras izstrādē
  - Formāls process - prognozējams un efektīvs – īpašā loma plānošanai
  - Sekojot konkrētai metodoloģijai ir jāizpilda visi tās norādījumi, kas var bremzēt produkta tapšanas tempu

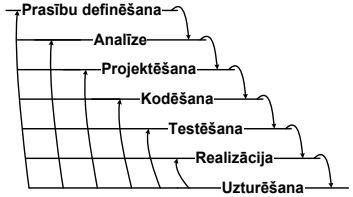
Lifecycle Models

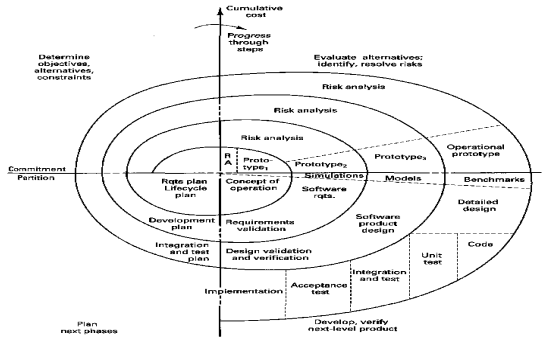
- Rapid prototyping
  - build something users can understand & assess
  - often focuses on the interface
    - e.g., Wizard of Oz studies
  - waterfall model follows the prototype
- Incremental (aka Evolutionary) development
  - incrementally expand the system
  - can be used to do a "phased delivery" to users
    - more expectation management needed!
    - build-and-fix revisited?
- Spiral Model
  - risk-driven approach
  - assess risks before each phase
    - what is the difficulty here?
  - re-assess in frequent cycles
- OO Models
  - more interaction between phases
  - more iteration within phases

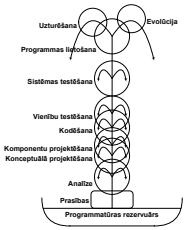
Programmatūras izstrādes procesu evolūcija

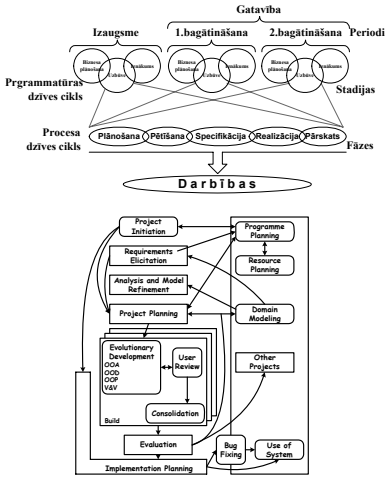
Code and Fix

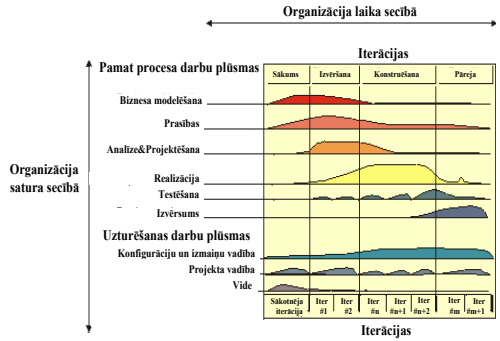


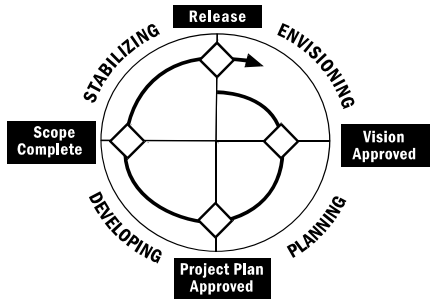












XP

Perils & Promises of Lifecycles

- Managers love 'em
  - defines a set of "deliverables"
  - sound bites – can tell upper management that "stage so-and-so is completed"
  - formally mandated by many government agencies
- Programmers find them inadequate
  - not representative of what happens in the "trenches"
  - customers can't adequately state requirements up-front
  - stages become intermixed w/ others
    - e.g., design reveals that part of specification can't be cost-effectively implemented

Spējā (Agile) programmatūras izstrāde (1/2)

- Spējās (viegla svara, veiklas, elastīgas) metodoloģijas mēģina sasniegt kompromisu starp stingri definētu izstrādes procesu un pilnīgi atbrīvotu no formalitātēm procesu
  - Mazāks dokumentācijas apjoms
  - Adaptīvs process (minimāla plānošana, nepārtraukta pielāgošanas apstākļiem)
  - Orientācija uz cilvēku (galvenais cilvēka dabas ievērošana, darbs priekā)

Spējā (Agile) programmatūras izstrāde (2/2)

- Programmatūras izstrādes procesa uzlabošanai jāsasniedz mērķi:
  - Maksimāli automatizēt kodēšanas procesu
  - Sistēmas analīzei un projektēšanai ir vajadzīgi talantīgi un radoši speciālisti
  - Radošs process ir grūti prognozējams – radošiem cilvēkiem ir jādod brīvība darbā

Disciplinēto un spējo pieeju salīdzinājums

Disciplinētās metodes	"Spējās" metodes
Stingri kontrolēti procesi. Izstrādātas stingras vadlīnijas, kurām ir jāseko	Procesi, kas organizējas "paši no sevis", par darbību secību izlemj komanda
Secīgi procesi ar noteiktu struktūru	Procesi savstarpēji pārklājās, nav noteiktas secības
Atkārtojams, universāls process	Katru reizi parādās jaunas iezīmes un vadlīnijām ir rekomendējošs status
Racionāls, iepriekšnoteikts, no mērķi vadīts process	Mainīgs, Apspriežams, uz kompromisiem balstīts

Spējā (Agile) programmatūras izstrāde

- Agile Software Development - spējā programmatūras izstrāde –
  - "informācijas sistēmu ātra izstrāde un ieviešana, ar pakāpeniskiem tuvinājumiem uzlabojot šīs sistēmas un iekļaujot tajās jaunākās lietotāju prasības. Programmas plānošanas gaitā sākotnējai specifikācijai nav jābūt visaptverošai, tajā jāietver tikai tās prasības, kas ir absolūti nepieciešamas. Prasību detalizācija un programmatūras tālāka attīstība tiek veikta tālākos attīstības posmos, tomēr, pirms programmatūra tiek izplatīta, tai jābūt darboties spējīgai un tajā nedrīkst būt kļūdu."

[www.termini.lv]

- "Mēs atklājam labākus programmatūras izstrādes veidus darot to un palīdzot citiem darīt to. Pateicoties šim darbam mēs esam sākuši cienīt:
  - Indivīdus un saskarsmi pār procesiem un rīkiem,
  - Darbojošos programmatūru pār saprotamu dokumentāciju,
  - Sadarbošanos ar klientiem pār kontraktu pārrunām,
  - Reaģēšanu uz pārmaiņām pār plāna izpildi.

- Proti, mēs novērtējam punktus pa labi, taču dodam priekšroku punktiem pa kreisi."

[Agile Manifesto]

"Agile Manifesto" principi

1. Augstākā prioritāte ir apmierināt klientu ar agru un nemitīgu vērtīgas programmatūras piegādāšanu.
2. Mainīgas prasības ir vēlamas pat vēlu projekta attīstībā. Elastīgais process to izmanto kā vienu no savām lielākajām priekšrocībām, ko klients var izmantot.
3. Piegādāt strādājošu programmatūru maksimāli bieži, sākot ar reizi divās nedēļās un beidzot ar reizi pāris mēnešos, priekšroku dodot īsākam intervālam.
4. Biznesa cilvēkiem un attīstītājiem ir jāstrādā kopā katru dienu, visu projekta izveides laiku.
5. Veidot projektus ar mērķtiecīgu personību palīdzību – dot viņiem darba vidi, atbalstīt viņu vajadzības un uzticēties, ka darbs tiks padarīts.
6. Vispraktiskākā un efektīvākā metode, kā nodot un uztvert informāciju, ir saruna aci pret aci ar darba komandu.
7. Strādājoša programmatūra ir primārā progresa mēraukla.
8. Elastīgie procesi nodrošina pastāvīgu attīstību. Sponsoriem, attīstītājiem un lietotājiem ir jāspēj saglabāt nebeidzami nemainīgu darba tempu.
9. Nemitīga uzmanība, kas pievērsta tehniski izcilībai un labam dizainam, paaugstina elastību.
10. Vienkāršība – māksla maksimizēt nepadarītā darba daudzumu – ir būtiska.
11. Labākā arhitektūra, prasības un dizains rodas pašorganizējošās komandās.
12. Ik pēc noteikta laika komanda spriež, kā darbu padarīt vēl efektīvāku, un tad noskaņojas un pielāgojas attiecīgi saviem spriedumiem.

Spējas metodoloģijas

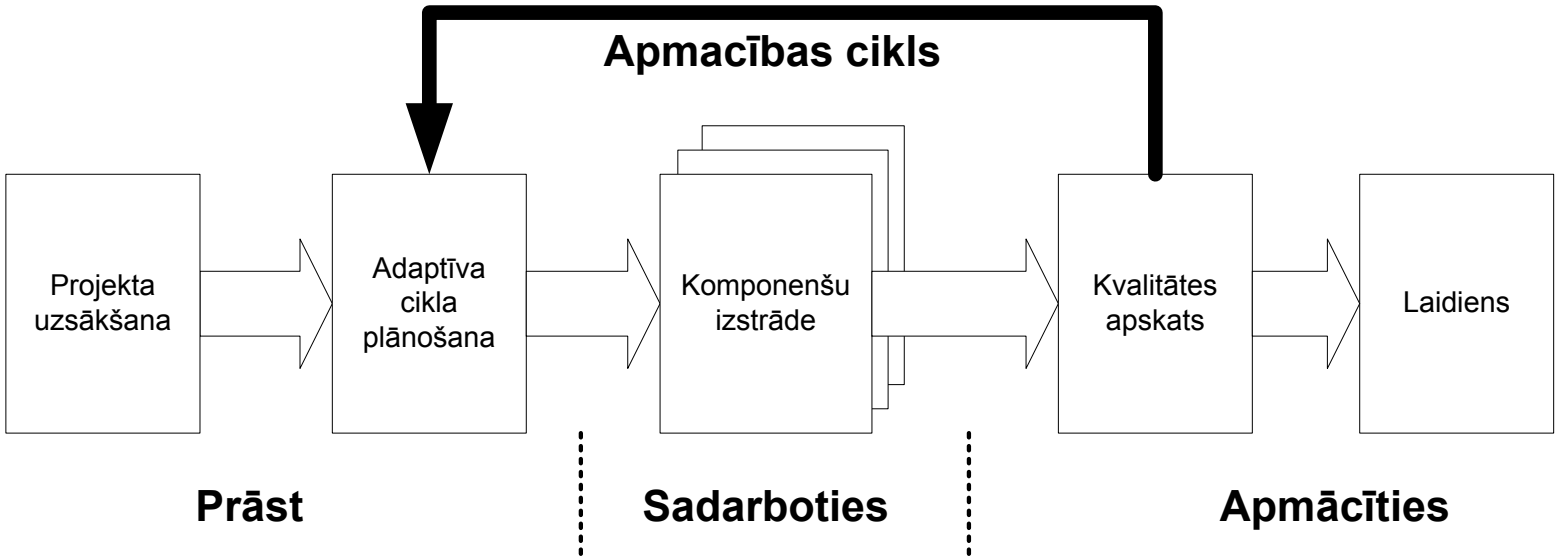
- Crystal (Alistair Cockburn: alistair.cockburn.us)
  - Piedāvā metodoloģiju kopumu
  - Projektu gradācija: aizņemto cilvēku skaits pret pieļauto kļūdu kritiskumu
  - Noskaidroja minimālo disciplīnas apjomu veiksmīgam projektam
  - Katras iterācijas nobeigumā var mainīt metodoloģiju

Spējas metodoloģijas

- Open Source (Erick Raimond, Karl Fogel)
  - Programmas kods ir vienīga dokumentācija
  - Projekta koordinators pārskata izstrādātāju piedāvājumus un ieraksta koda repozitorijā
  - Projekts var būt sadalīts moduļos, tad katram modulim savs koordinators
  - Lielākā daļa tiek tērēta kļūdu meklēšanai un labošanai

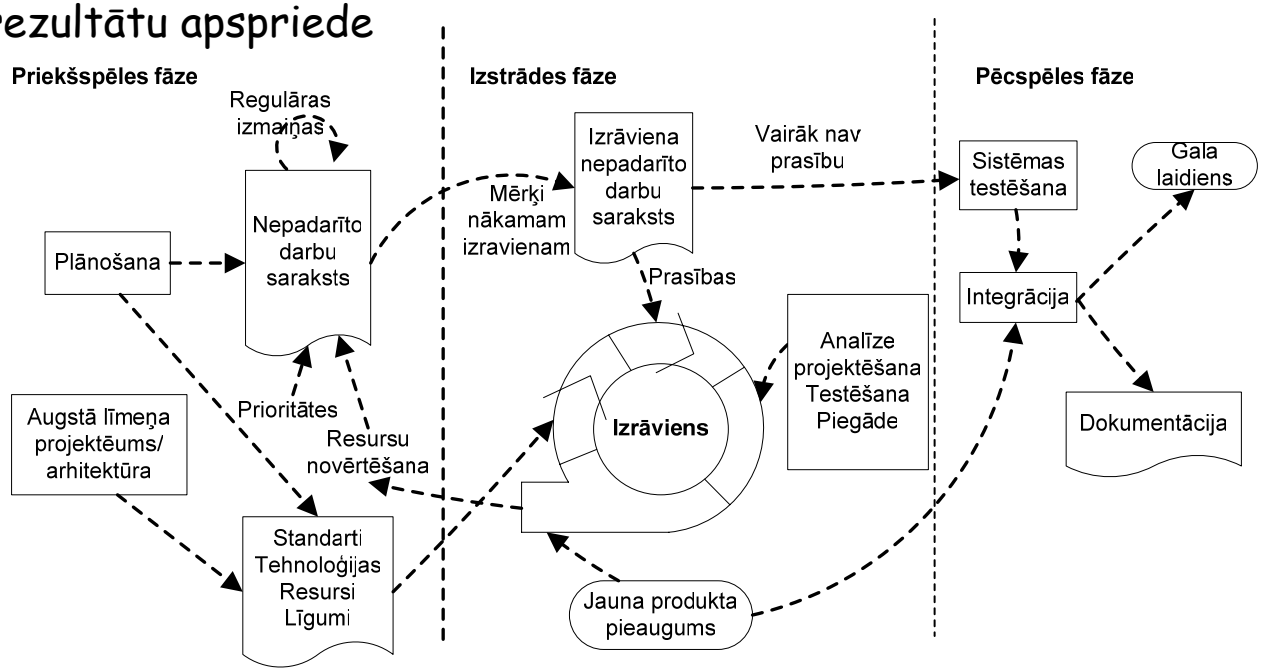
Spējas metodoloģijas

- Adaptive Software Development (2000: Jim Highsmith; www.adaptivesd.com)
  - Haosa teorija (nav detalizēta izstrādes apraksta)
  - Trīs pārklājošas fāzes: domāšana, sadarbība, apmācība
  - Plānošana ir paradokss: parastajā plānošanā - novirzieni ir kļūdas, adaptīvajā plānošanā – novirzieni ir pareizie risinājumi



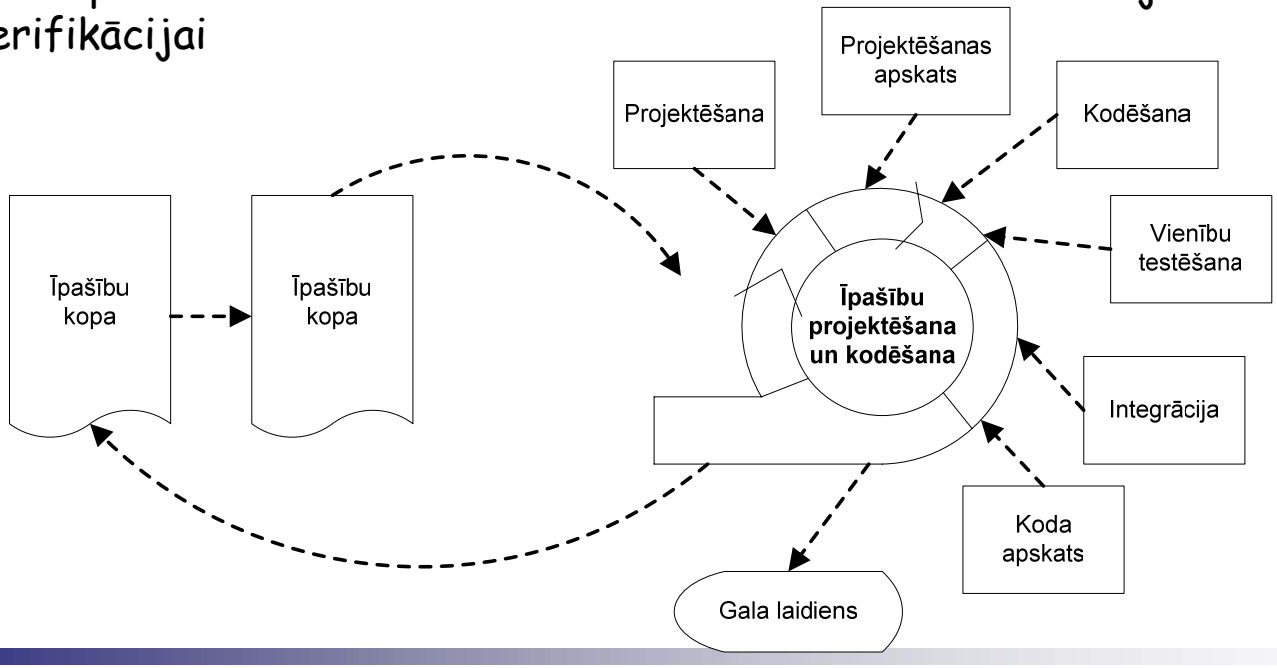
Spējas metodoloģijas

- SCRUM (1986, 1995: Ken Schwaber, Mike Beedle; controlChaos.com)
  - Objektorientēta izstrāde
  - Projekts dalās iterācijās (sprintos) 30 dienas katra – viena sprinta gaitā tiek realizēts noteikts sistēmas funkcionēšanas bloks ar nosacījumu, ka prasības netiek mainītas
  - Katru dienu komandā savācas uz 15 minūtēm (scrum), kurā notiek problēmu un rezultātu apspriede



Spējas metodoloģijas

- Feature Driven Development (2000: Jeff De Luca, Peter Coad, Jos Palmer)
  - Īsas iterācijas (divas nedēļas)
  - Pieci procesi: kopēja modeļa izstrāde, sistēmas īpašību saraksta sastādīšana, katras īpašības apstrādes plānošana, katras īpašības projektēšana, katras īpašības konstruēšana
  - Katrs process tiek sadalīts uzdevumos un ir noteikti kritēriji rezultātu verifikācijai



Spējas metodoloģijas

- eXtereme Programming (1999: Kent Beck, Martin Fowler; extremeProgramming.org)
  - 4 pamata principi: Komunikācija, vienkāršība, atpakaļ ejoša saite (feedback), drosme; 12 prakses (īpašas ir user stories, pair programming)

Extreme Programming (XP)

- Kent Beck

- "Extreme Programming turns the conventional software process sideways. Rather than planning, analyzing, and designing for the far-flung future, XP programmers do all of these activities a little at a time throughout development."
- IEEE Computer October 1999

- XP Book Series from AW

What is XP ?

- XP is giving up old, ineffective technical and social habits in favor of new ones that work.
- XP is fully appreciating yourself for total effort today.
- XP is striving to do better tomorrow.
- XP is evaluating yourself by your contribution to the team's shared goals.
- XP is asking to get some of your human needs met through software development.

by Kent Beck

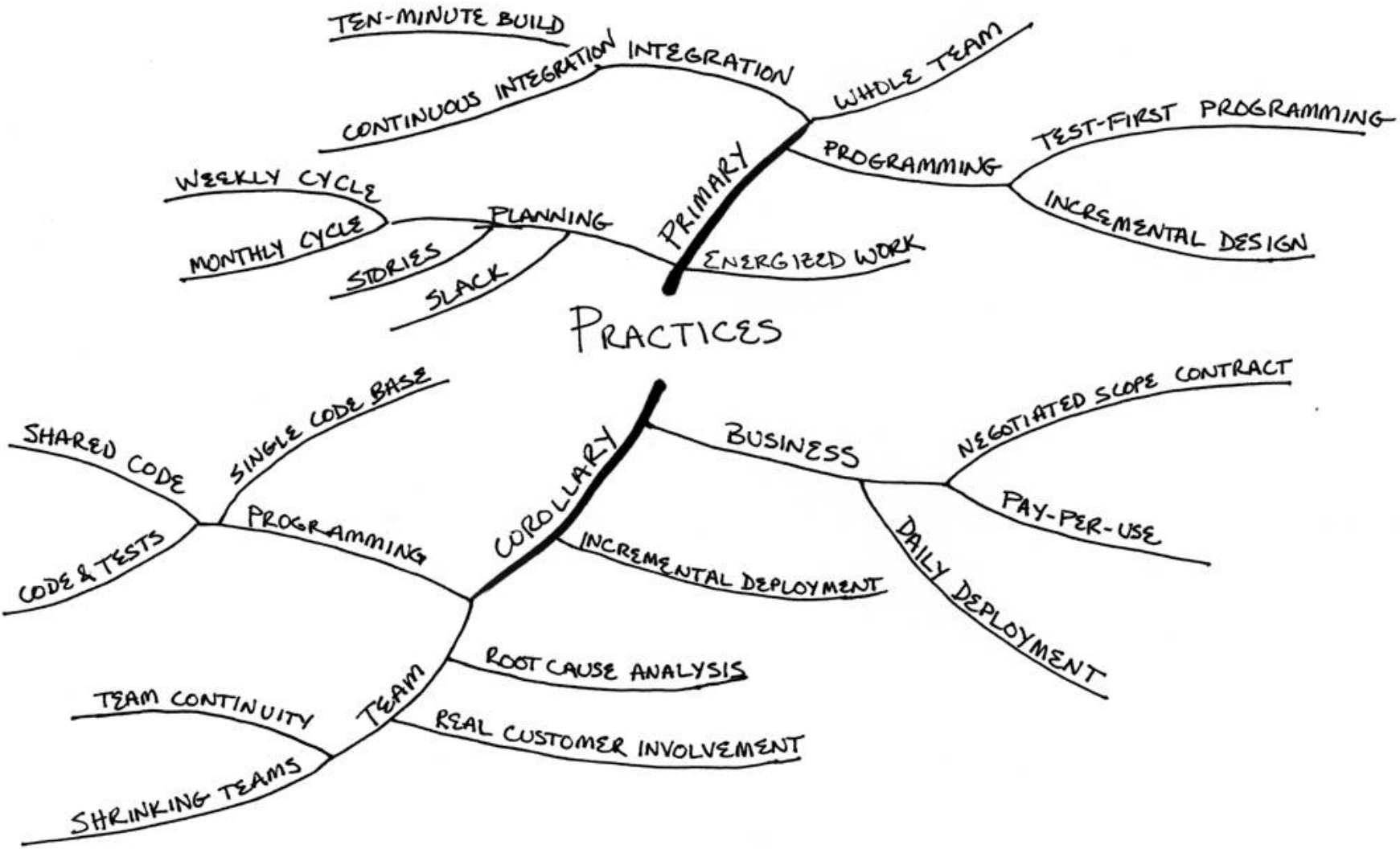
Why XP is extreme?

- XP is extreme because it taking best software engineering practices to extreme

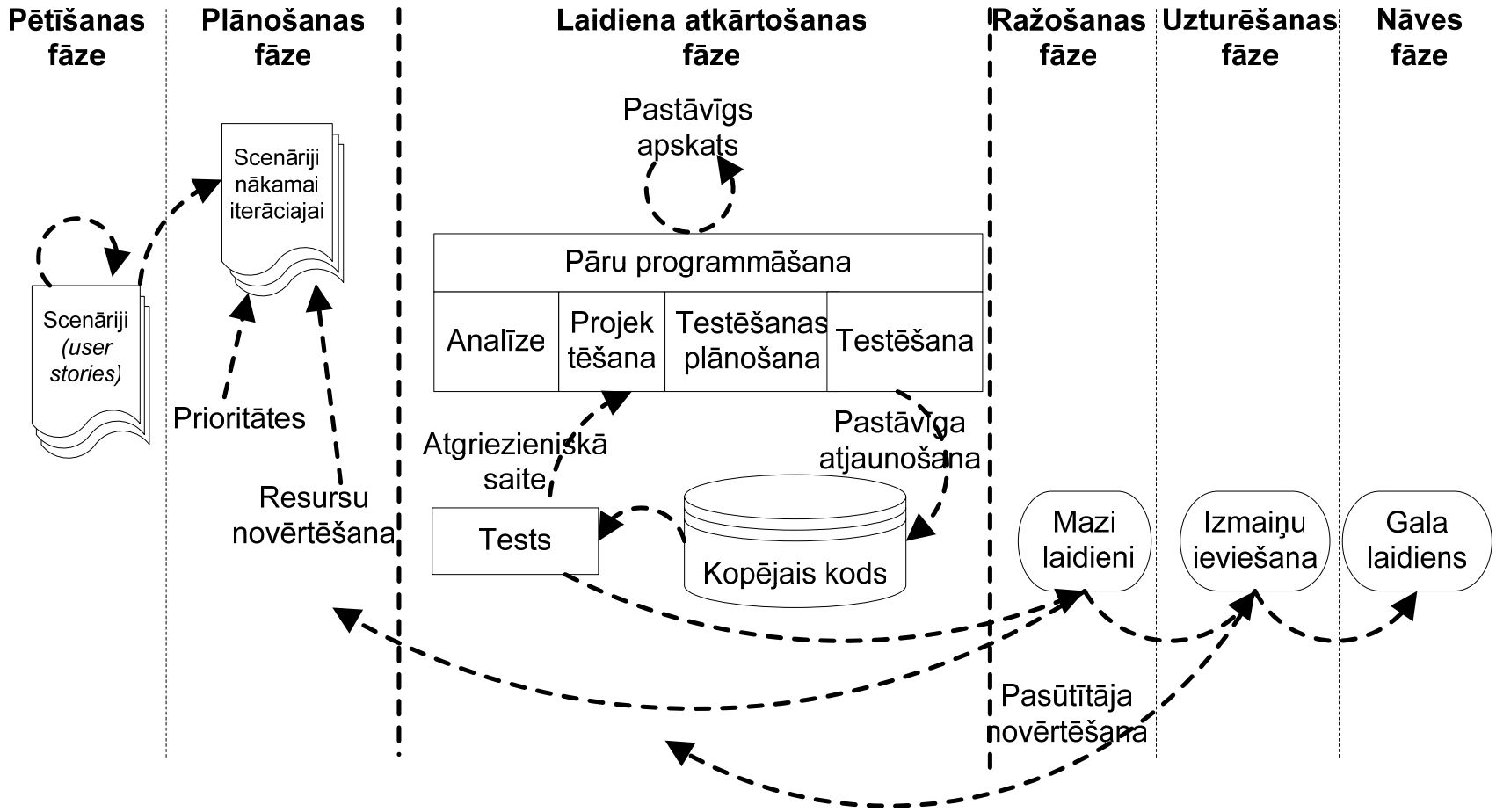
XP Practices

- Practices are the core of XP
- Practices should be used for purpose
- Practices are tightly related and support each other

XP Practices cont.



XP dzīves cikls (Beck, 1999)



Plānošana

- Plānošanas fāzes XP prakses ietver projekta kopēja laika un naudas budžeta novērtēšanu, šeit, kā arī tās ir vairāk orientētas uz projekta vispārējo vadību.

Plānošanas prakses (1., 2., 3./8)

- **Lietotāja stāsti** - cieša sadarbošanās starp programmētājiem un klientu. Programmētāji informē klientus par laiku, kas būs nepieciešams, lai realizētu stāstus. Klienti, savukārt, saņemot šādu informāciju, izlemj, kurā no iterācijām realizēt katru stāstu.
- **Laidienu plānošana** - Paša pamatā laidiena plāns nosaka to – kādas funkcijas (Lietotāja stāsti) tiks realizētas nākošajā programmatūras laidienā un arī to, kad tas notiks (konkrēts datums).
- **Mazi laidieni** - tiek realizēta iteratīva programmatūras izstrāde, kas ļauj piegādāt klientam programmatūru pēc iespējas ātrāk, vispirms klientam tiek piegādāta programmatūra, kas realizē galvenās funkcijas

Plānošanas prakses (4., 5./8)

- **Projekta ātrums** - projekta ātrums ir koeficients, kas parāda attiecību starp ideālo izstrādes laiku un reālo izstrādes laiku
- **Iterācijas / Iterāciju plānošana** - Programmatūras izstrādes process tiek sadalīts nelielās iterācijās pirms katras no kurām notiek iterācijas plānošana. Tipisks iterāciju ilgums ir 2-3 nedēļas. Iterāciju ilgums tiek uzturēts konstants un pēc šo iterāciju rezultativitātes var viegli spriest par projekta izstrādes ātrumu un plānot programmatūras laidienus.

Plānošanas prakses (6., 7., 8./8)

- **Personāla rotācija / Pāru programmēšana** - Viena no projekta problēmām ir "šaurās vietas" izveidošanās kāda koda zonā, ko pārzina tikai viens vai daži izstrādātāji. Ja šāds izstrādātājs pēkšņi pamet projektu vai arī tiek aizskrauts ar darbiem, tad progresa trūkumus šīs programmatūras daļas attīstībā var viegli nobremzēt visa projekta kopējo attīstību.
- **Sanāksme kājās stāvot**
- **Labojam XP** – nav stingras metodoloģijas ir rekomendāciju komplekts, ko var kombinēt, pielāgojoties projekta specifikai

Projektēšana

- Sistēmas projektēšanai XP tiek pievērsta tikai tik liela loma, lai nepieļautu tai traucēt izstrādes gaitu

Projektēšanas prakses (1., 2., 3./6)

- **Vienkāršība** - uzsvars tiek likts uz pēc iespējas vienkāršoto projektēšanu. Sarežģītība, sarežģītības pēc nav pieņemama un nevajadzīgi elementi, koda gabali tiek izmesti.
- **Sistēmas metafora** - sistēmas izstrāde ir balstīta uz metaforu apkopojumu, ko saprot gan programmētāji, gan klienti. Citiem vārdiem sakot, sarunājoties savā starpā abas puses lieto terminus, kuri ir saprotami pārāk neiedziļinoties niansēs.
- **CRC kartes** – Wirfs-Brock klašu attiecību un atbildību pieraksta formāts

Projektēšanas prakses (4., 5., 6./6)

- **Triecienrisinājumi** - Triecienrisinājumi ir kādas tehnoloģijas izmēģināšana testa veidā, lai atrisinātu kādu svarīgu tehnisku vai dizaina problēmu un uz šī risinājuma pamata pieņemtu lēmumu, novērtētu lietotāja stāsta sarežģītību vai novērstu kādas potenciālas tehniskas problēmas risku.
- **Nepievienot funkcijas par agru**
- **Refaktorings** - sistēmas restrukturizēšana, aizvācot nevajadzīgos koda gabalus, vienkāršojot darbību, uzlabojot komunikāciju un elastību.

Testēšana

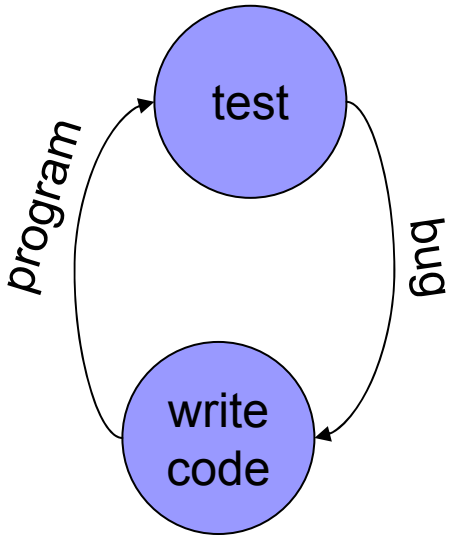
- Programmatūras izstrāde ir balstīta uz testiem. Vienību (objektu) testus ievieš pirms koda un veic nepārtraukti. Klienti sastāda funkcionālos testus.

Testēšanas paņēmieni (1., 2., 3./3)

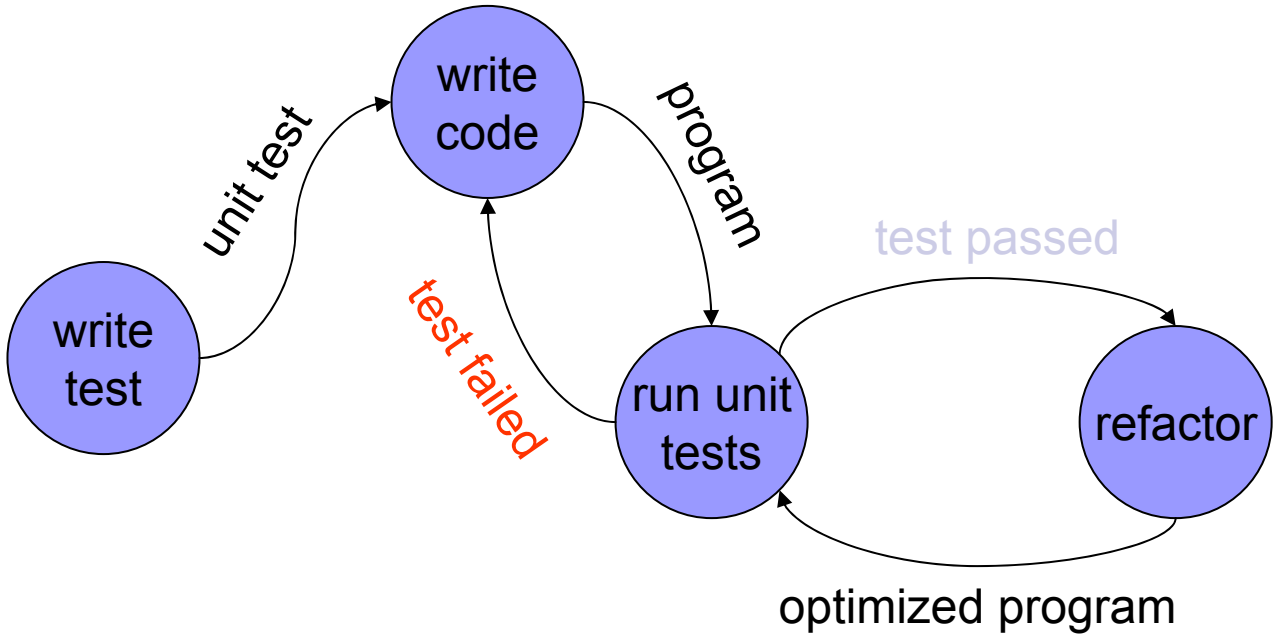
- **Vienību testi** – vienību testi tiek rakstīti pirms pašu vienību uzrakstīšanas un tiek integrēti vienotā testu kompleksā, kas ļauj pilnīgi automātiski veikt visu testēšanas procesu.
  - Vienību testi – laidiena indikators – vienību testi tiek pielietoti laidena gatavības pārbaudei.
- **Kļūda = Tests** – kopēja repozitorijā atrasta kļūda nozīmē, ka vienības testi nebija pietiekoši pilnīgi uzrakstīti. Lai to labotu ir jāuzraksta testi, kas parāda atrasto kļūdu, un integrējoties ar kopējo testēšanas sistēmu nedod aizmirst par atrasto problēmu.
- **Pieņemšanas testi** - no lietotāja stāstiem atvasināti un ar pasūtītāju saskaņoti testi, kas ir orientēti uz programmas funkcionalitāte attiecībā pret gala lietotāju.

Coding Styles

Traditional approach



XP approach



Kodēšana

- Kodēšanas process, kas tradicionālajās metodoloģijās tiek pazemināts līdz "melnajam darbam" vai pat aizstāts ar koda ģenerēšanas rīkiem un algoritmiem, XP gadījumā tiek pasludināts par galveno projekta izstrādes fāzi.

Kodēšana (1., 2., 3./9)

- **Klients vienmēr ir pieejams** – klientam ir jābūt pieejamam no komandas puses katru brīdi.
- **Kodēšanas standarti** - tiek uzsvērta komentāru lietošana kodā – komunikācija caur kodu. Tā pat ir jāievēro arī "labā stila" kodēšana.
- **Vienības tests pirms koda** - XP iesaka rakstīt vienības testus pirms paša vienības koda rakstīšanas, lai varētu vienības testus izmantot par kritēriju, ka vienība ir pabeigta. Ja visi vienības testi, kas ir uzrakstīti pirms pašas vienības, sāk strādāt pēc vienības uzrakstīšanas, tad vienība ir pabeigta.

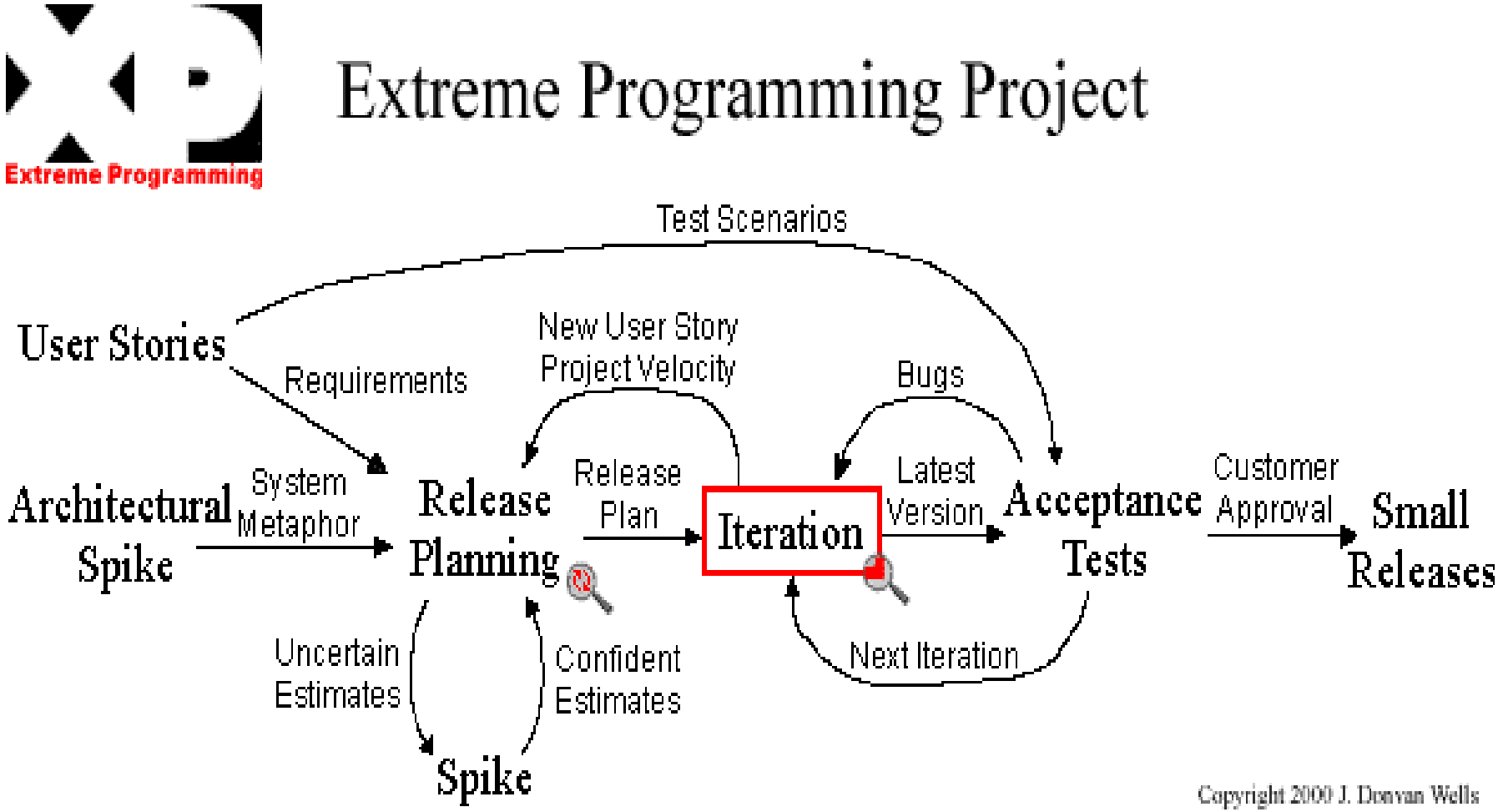
Kodēšana (4., 5., 6./9)

- **Secīga integrācija** - integrāciju vienlaicīgi var veikt tikai viens programmētājs, kas šīm nolūkam pielieto vienu, unikālu fizisku objektu, piemēram, var izdalīt vienu datoru un atļaut veikt integrāciju tikai no šī datora.
- **Bieža integrācija** – neatlikt sistēmas integrāciju uz pēdējām nedēļām, bet katru dienu integrēt sistēmā šajā diena izveidoto koda gabalu
- **Kolektīvas koda īpašuma tiesības** - jebkuram programmētājam ir tiesības mainīt kodu jebkurā programmas vietā. Vienību testi ļauj programmētājam, kas maina cita programmētāja izstrādāto objektu, pārliecināties, ka viņa darbības neizjauc objekta funkcionēšanu.

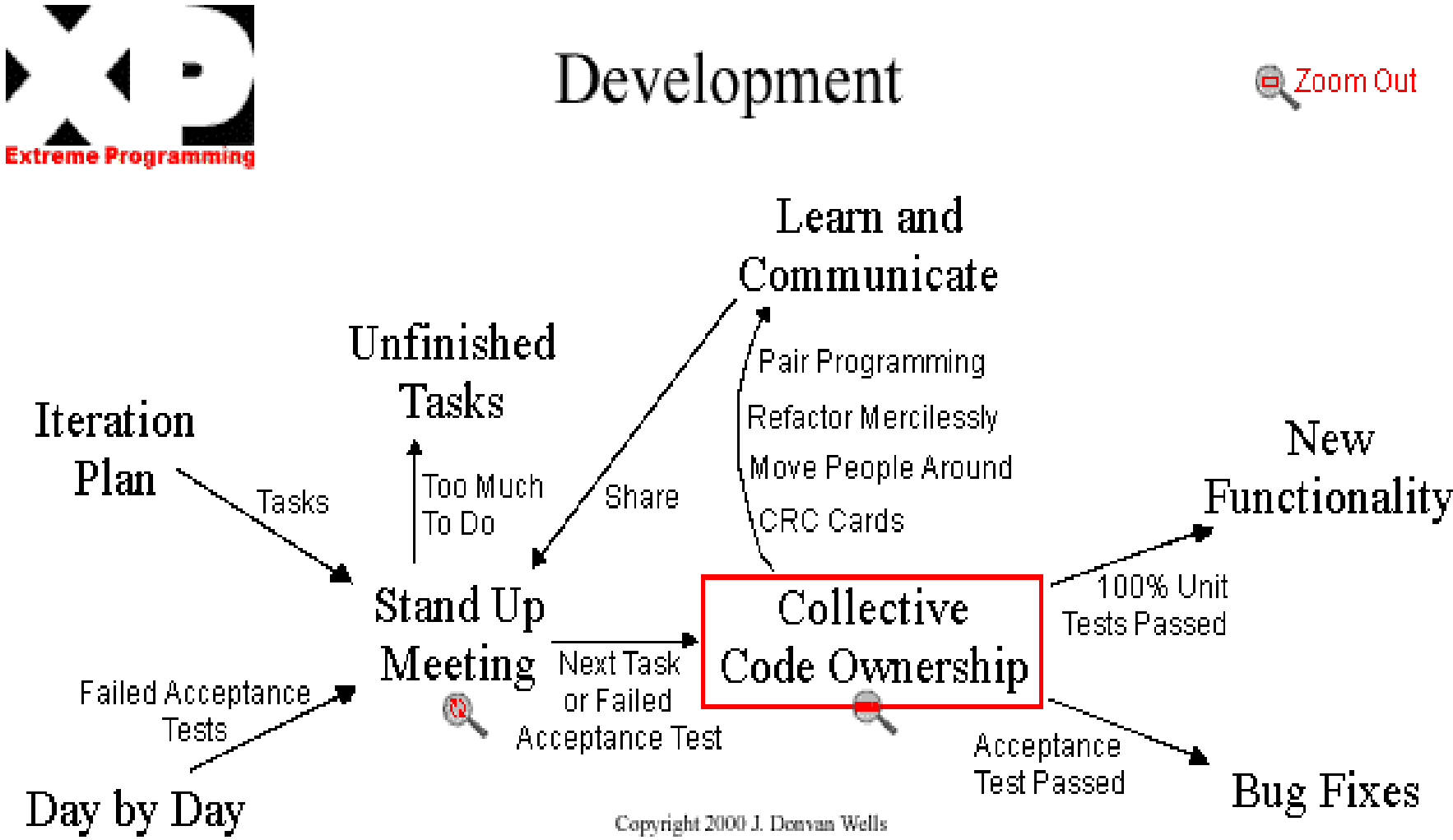
Kodēšana (7., 8., 9./9)

- **Optimizē vēlāk**
- **Bez pārslodzes / 40-stundu darba diena** – nav pieļaujamas divas vai vairāk nedēļas, kurās ir bijis jāstrādā papild stundas. Ja tā tomēr notiek – to uzskata par problēmu, kuru vajag risināt.
- **Īsas iterācijas/relīzes** – no projekta uzsākšanas līdz tā ievadīšanas produkcijas fāze paiet divi vai trīs mēneši. Jaunas versijas pēc tam tiek izlaistas daudz biežāk, pat ik dienu.

Ekstremālā pieeja sistēmas izstrādē



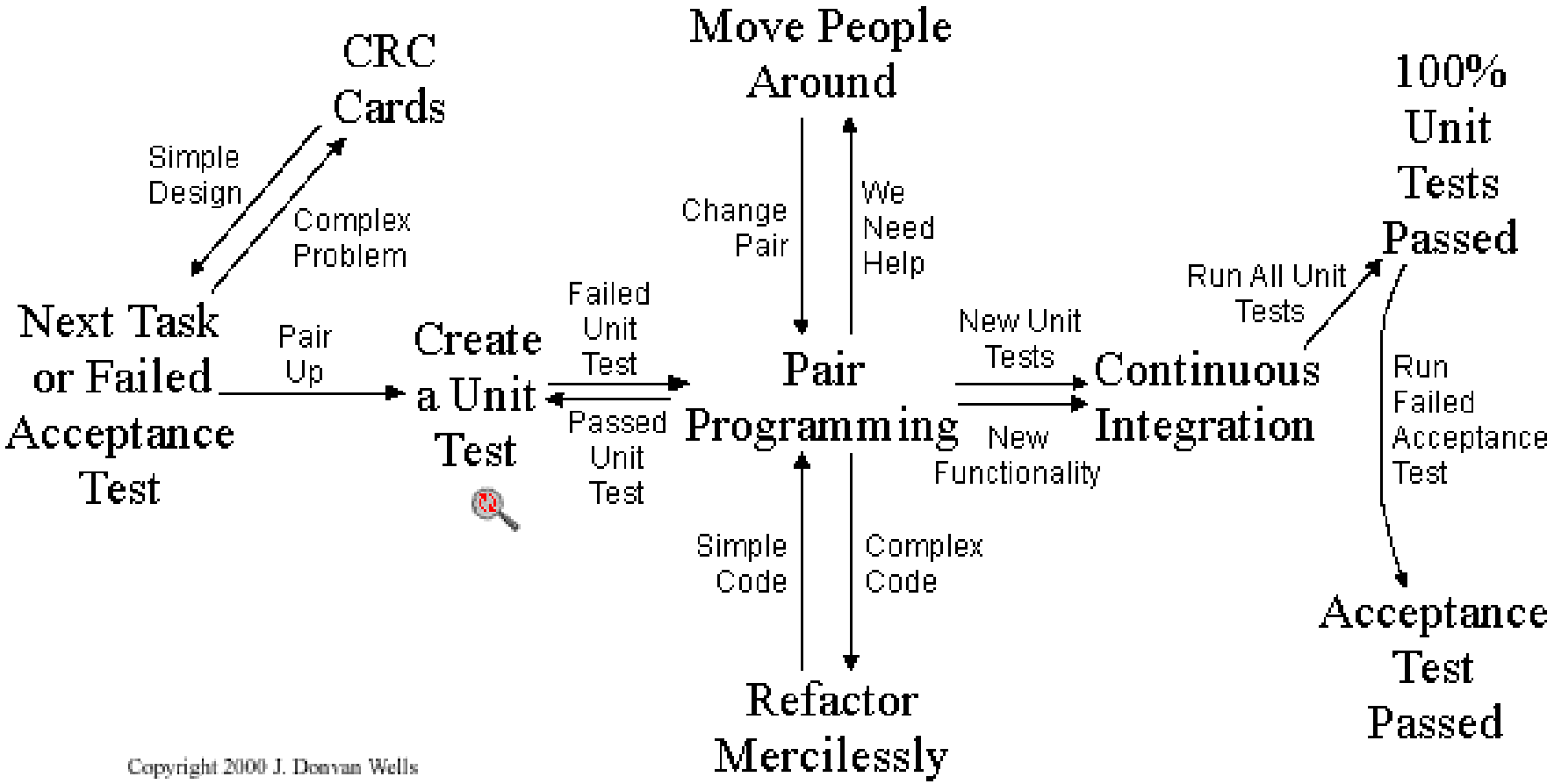
Ekstremālā pieeja sistēmas izstrādē



Ekstremālā pieeja sistēmas izstrādē

Collective Code Ownership

Zoom Out



Places to start with

- Internet
  - groups.yahoo.com/group/extremeprogramming/
  - http://www.extremeprogramming.org/

- Books (DC++, EMule, www.amazon.com, local book stores)
  - Extreme Programming Explained, Second Edition: Embrace Change, Kent Beck with Cynthia Andres
  - Extreme Programming Applied: Playing to Win, Ken Auer and Roy Miller
  - Planning Extreme Programming, Kent Beck and Martin Fowler
  - Refactoring, Martin Fowler