Global Product Development

Thoughts and experiences about building software globally effectively

Hannu Kokko, 8 December, 2019

Did you know this about Elisa as a builder of unique digital services

96% of the customers who moved to our application service have ordered more services from us

1

contact point to lead a team built for you

+100

Digital services in application service and development

200

developers at your service

24/7

Support

1 wk

Fast start with one week sprint

1

All followup and monitoring from single point

Questions in the start

- Agile
- Kanban
- Continuous integration
- DevOps
- Team / project sizes

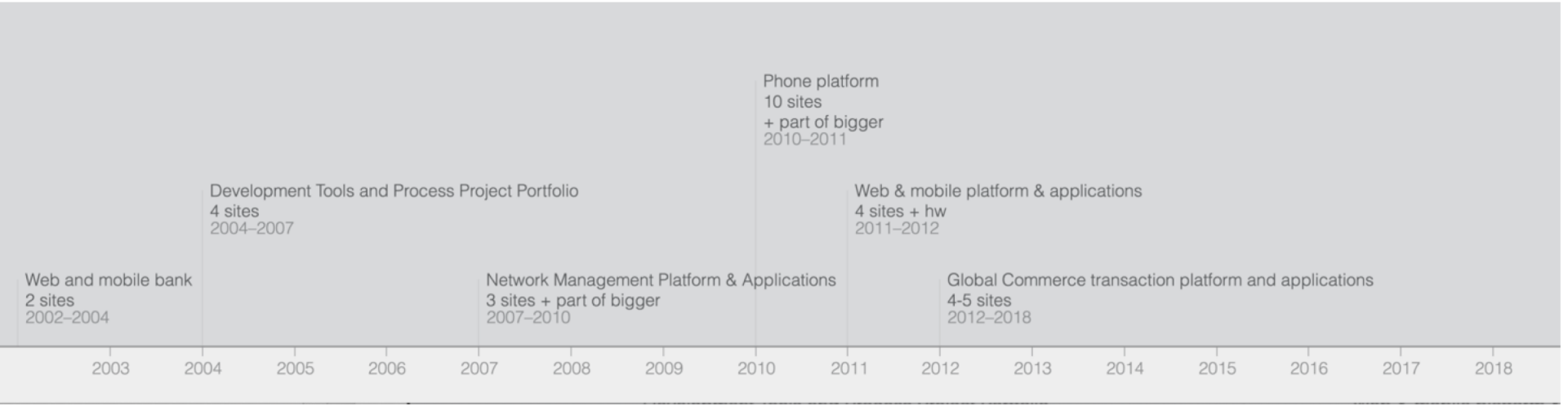
Distributed Product Development

Fragments In My Story

Building, improving and leading distributed product development organisation in agile over long term is multifaceted challenge.

Ways of working together and emergence of common culture over geographical borders, leading the development and setting and following common goals are all important factors.

Over the journey agile, lean, team of teams, estimation practices, CI, common situational view and the usage of both technology and partners have developed and been elements of team success.



Stuff I've done since 1983

Unic Intime accounting package
Kultaraha web bank
Corporate bank systems (Smalltalk)
Wellmate beta (Android, Symbian, Web, hardware)
Merlin Production Pipe (company internal CI environment)
Basware Portal
OPNET payment system
Customer projects to multiple customers in eight computer languages, Java since 1997+
Basware Commerce Network
Nokia N9 (MeeGo phone - application framework, desktop)
NSN OES Middleware Suite (J2EE Network Management Platform)
Meridea Financial Suite (J2EE Web and mobile banking application)

What skills do you need in software and ICT nowadays and why? How can you gain them

Tech – changes rapidly

Full stack

Programming languages: Javascript, Python, Java, others

Mobile application development skills: iOS, Android

UX, AI, Data science

Mindset – more constant

Agile…

Willingness to **constantly learn** and **experiment**

Constant **curiosity** about new topics

Capability to work in **self organized** teams

Why: things change very quickly

More than one path

Apprenticeships

Summer trainees

Hobby projects

Open source

MooC

Universities

Reading widely

Networking

Mentoring

Software & ICT skills

Diversity in teams

Experience, short to have seen it all

Females, males, any other

Global, Local

How:

Grow - different doing paths. Think long term – rotate

Support and work together with learning institutions

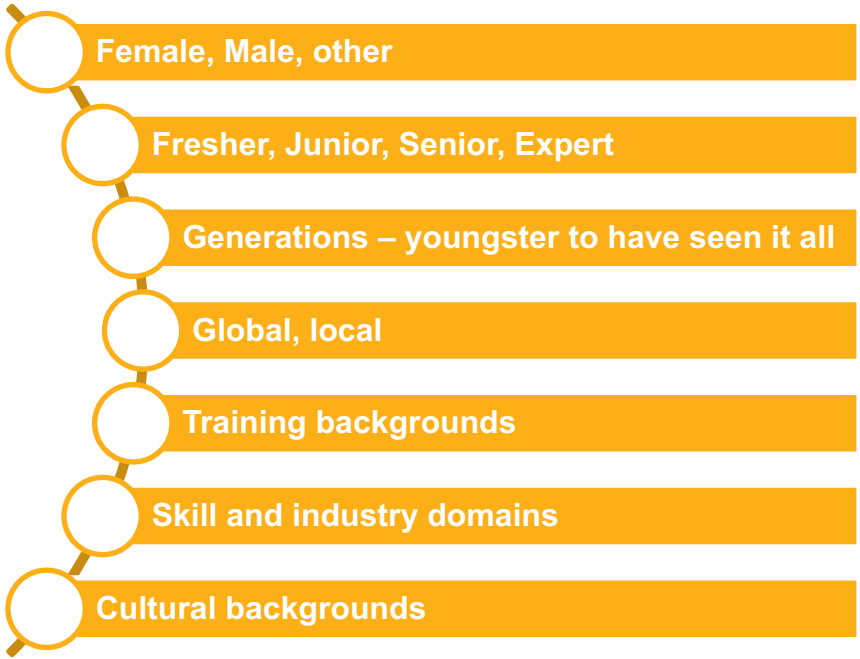
Hire young – Hire existing talent – the balance

What do companies need, how do they get more skilled workers?

Diversity in teams

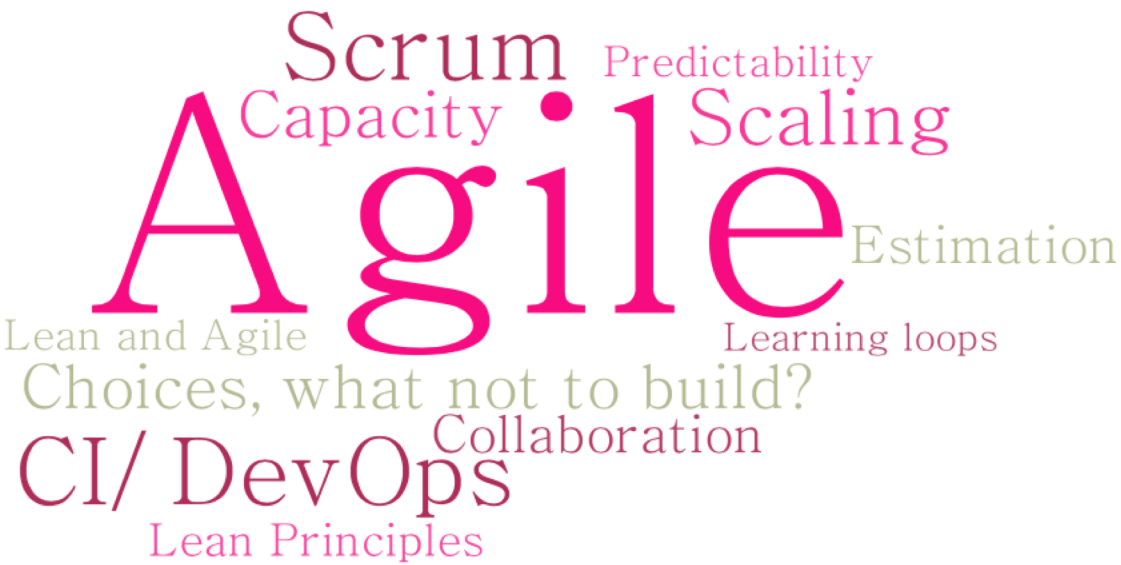
Diversity of all kinds in the teams helps build the solutions the diverse world deserves and needs.

Co-operation with university helps expand the diversity.



Aspects of building software globally effectively

- Based on my experience across multiple companies - not a specific company – not any company view



The most important thing: People

Hire well

Involve teams in hiring

Take good care of the people hired

Build for long term

Culture of learning

Culture of working together

Be an example

Some things I have found useful…

- Learn – ship – learn. Ship early.
- Team of teams – flow, common situational view and common team targets, scaling and dependencies need attention and work constantly
- Understand the customer
- Everybody in the team needs to understand the problems AND the solutions.
  - Keep quality high constantly
  - Open eyes to the world
  - Always think of multiple different ways to build
  - Think about the future when building the present
- Listen…

Global Product Development Thoughts

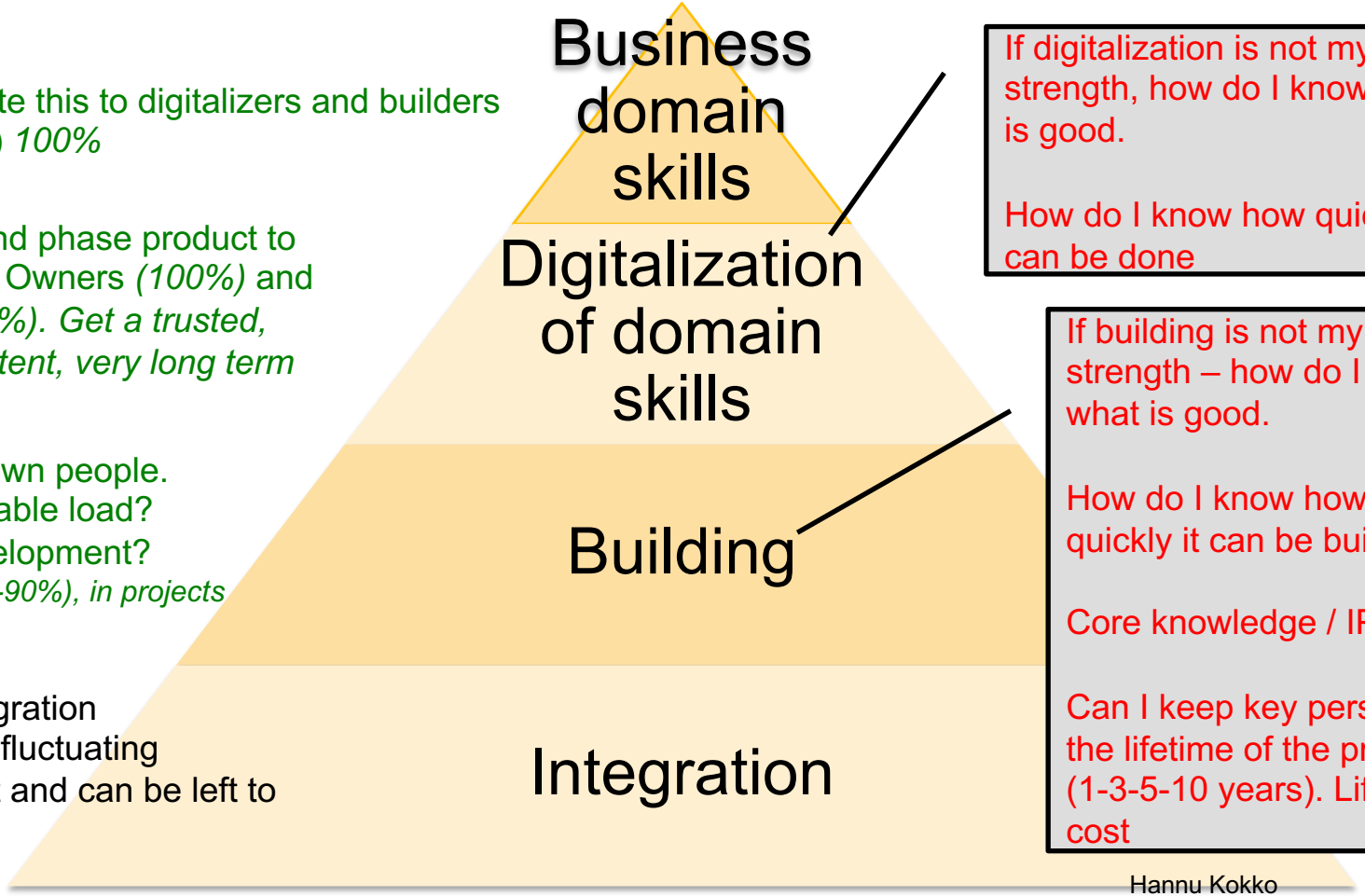
What are your company core competencies? What they should be?

Can I communicate this to digitalizers and builders (Business Owner) *100%*

Can I structure and phase product to builders (Product Owners *(100%)* and Architects *(70-90%). Get a trusted, extremely competent, very long term partner*

Can I scale with own people. Output? Cost? Stable load? Competence development?
In product dev (70%-90%), in projects can be less

Often use of integration technologies are fluctuating project by project and can be left to a partner.



If digitalization is not my strength, how do I know what is good.

How do I know how quickly it can be done

If building is not my strength – how do I know what is good.

How do I know how quickly it can be built.

Core knowledge / IPR.

Can I keep key persons the lifetime of the product (1-3-5-10 years). Lifetime cost

Agile Manifesto

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

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

That is, while there is value in the items on
the right, we value the items on the left more.

Working practices

- Agile is useful in most cases
- Scrum and Kanban when followed are pretty good ways of working
- Short iterations are better than long ones. 1 week is ok for almost anything.
- Build shippable user value not technical stories
- Build good automated tests
- Integrate several times a day. Ship that often if you can
- Think about architecture and incurring technical debt
- Do the simplest thing that can possibly work

Agile: Nokia Test from 2007 is still a pretty good but basic canary

First, are you doing Iterative Development?

- Iterations must be timeboxed to less than 4 weeks
- Software features must be tested and working at the end of each iteration
- The Iteration must start before specification is complete

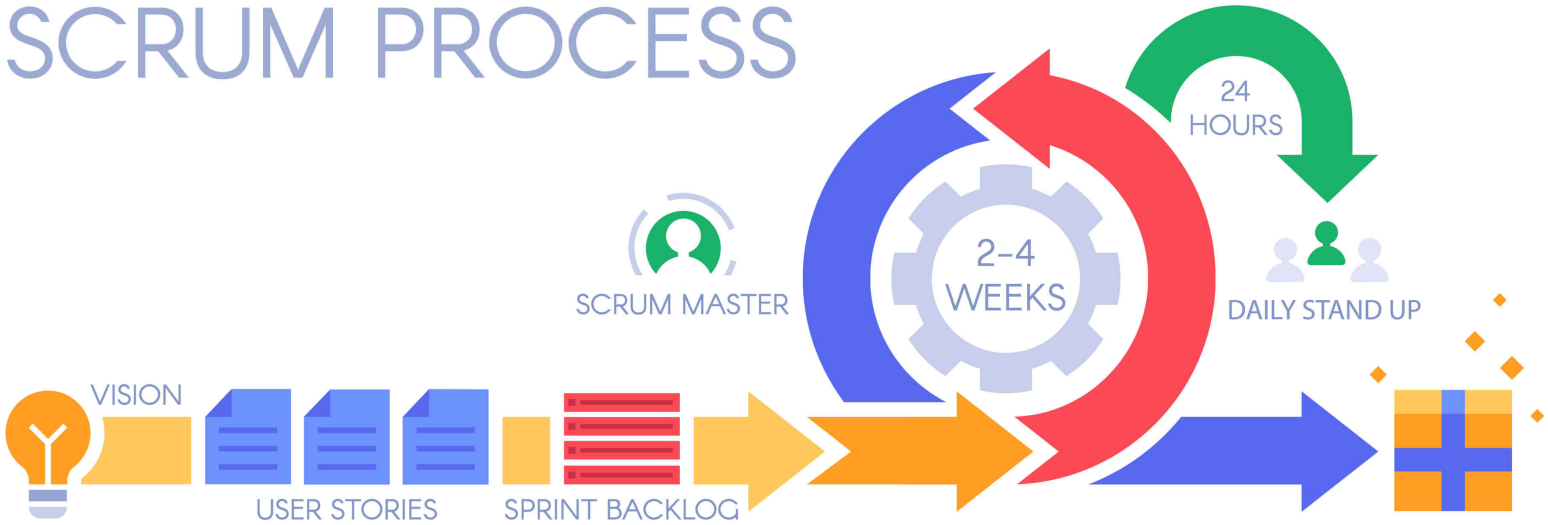
The next part of the test checks whether you are doing Scrum (in Nokia's opinion):

- You know who the product owner is
- There is a product backlog prioritized by business value
- The product backlog has estimates created by the team
- The team generates burndown charts and knows their velocity
- There are no project managers (or anyone else) disrupting the work of the team

Add these at least:

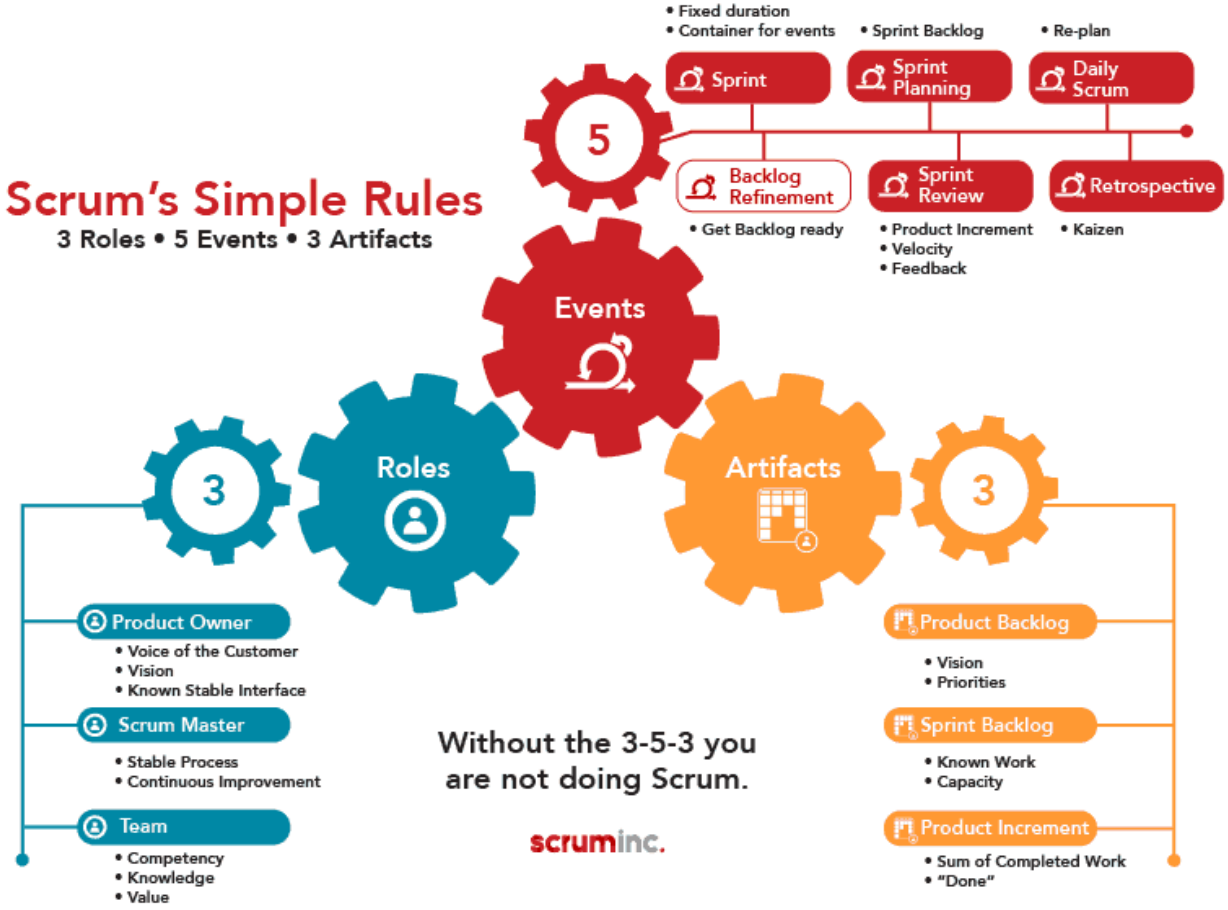
- Timebox is one (preferably) or two weeks
- Are you doing retrospectives at end of each iteration + from the retro during next sprint. **doing at least one item**
- Do all the developers check in their code at least once a day and integrate it in CI
- Are you shipping to production at least once per iteration

Scrum



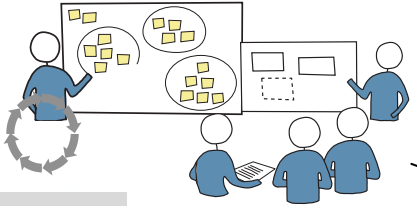
https://www.ntaskmanager.com/blog/newbies-guide-to-scrum-project-management-101/

Scrum

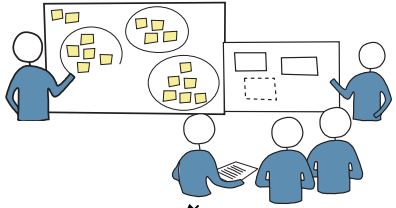


https://www.scruminc.com/the-3-5-3-of-scrum/

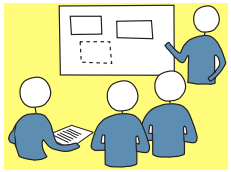
Basics in Place



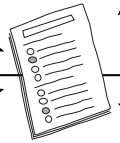
Roadmapping



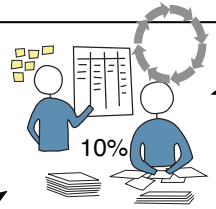
Release planning



Release info session



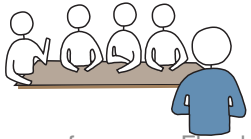
Backlog



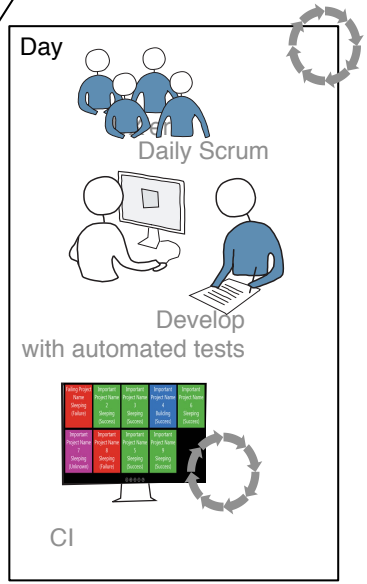
Constant Grooming for following sprints

User stories Definition of Ready

Team Planning session



<Scrum of scrums> Flowdock



Daily Scrum

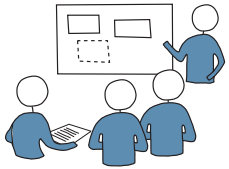
Develop with automated tests

CI

Sprint demo

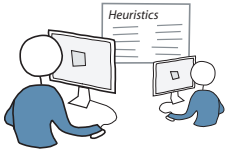
Retrospective

DevGrooming



Release retrospective

Release criteria checklist



Release test & deploy

Continuous Integration (CI) is a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early.

By integrating regularly, you can detect errors quickly, and locate them more easily. (from ThoughtWorks)

- **DevOps is the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support.**
- A primary corollary to this is that part of the major change in practice from previous methods is
- **DevOps is also characterized by operations staff making use many of the same techniques as developers for their systems work."**
  - **From AgileAdmin blog**

Single truth of the working software. Tests need to be good. Needs to be fast.

Experiences from being PO for CI environments over 11 years

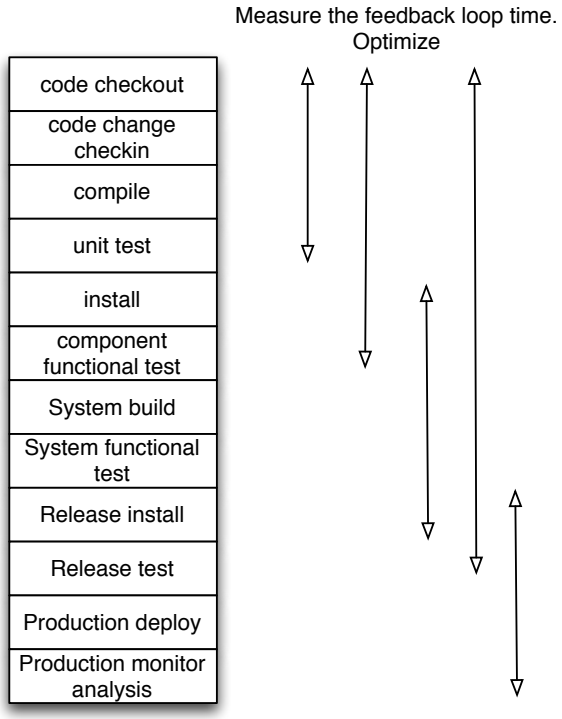
- CI environment should be developed in an agile way
- Pure CI can work even in very large environment
- Product owner is useful…
- Faster feedback times can be achieved with performance engineering techniques
- **Faster feedback time** and **constant visibility** brings more predictability and productivity to software development
- Context is important
- Bringing old codebases to CI – lot of work

Era of the CI, CD and DevOps 2007-2018+

Product 1: "installed"	Product 2: "embedded"	Product 3: "web/mobile/hardware - cloud"	Product 4: SaaS / Cloud
Large code base	Extremely large code base – open source contribution & development – basically a Linux distro + applications	Compact code base. Hardware development	Multiple old code bases and a new one.
New development and refactoring. Maintenance	New development + hardware development	Refactoring in the large + applied research and new development	Refactoring in the large, new development + maintenance
Mostly server code – Java & friends	Embedded code C++	Java, JavaScript, C++, HTML5 – multiple phone OS, multiple browsers	Java, Scala, JavaScript, HTML5. Mostly server code. Multiple databases (SQL, NoSQL). For all new development : startup stack
Dozens of scrum teams. Multiple countries, multiple locations	Many dozens of scrum teams. Multiple countries, multiple locations	9-10 scrum teams. Multiple countries, multiple locations	Between 10-20 scrum teams. Multiple countries, multiple locations

Learnings from Product 1-4

Faster feedback loops to learn faster
Automate, Measure, make visible, optimize



Make visible

Parallelization by flexible use of **development cloud and QA cloud cost effective** way of getting faster

Hannu Kokko

Learnings from Product 1 (“installed”)

Use performance engineering to get faster – small change every sprint

Wide impact tuning
-Fast builds – less changes – more green
Parallelization of test runs
-Clearcase to subversion
-Preinstalling as much as possible
-Removal of randomness
-Compilation in memory
-Installation starting parallel with system build

Focused impact tuning
-More memory and cpus
-Parallelize builds
Replace 3rd party test libraries
Reduce/remove timeouts in tests
Select different tests
Refactor code & components
-Tune the network & software
-Tune the database

Process learnings for succeeding with large scale CI

Overall

- Active change agent/sponsor/owner needed
- Continuous, persistent improvement – use agile
- **invest in local coaching during deployment**
- Actionable radiators everywhere.
- Integrate the visibility. History is useful
- Have dedicated team but do not separate it to different organisation. Super close co-op is the key.
- Eliminate with hard hand all excuses for not automating
- Anything can be made much faster
- Keep builds & tests fast
- Do not hide long in branches.
- Feature flags are useful (product 4)

For developers and program

- Small batch sizes – fast feedback -> increased productivity
- Unlikely to prove at component level that integration will work
- Mindset change – grow the code.
  - Do not hide in branches: Commit constantly – every day.
- Program rules – “fix the build”

Lean principles – speed

- Constantly question the necessity of steps, length of steps, parallelization and serialization of steps in the overall (**#CI**) system

- Make it a habit of being a bit faster every day. Refactor in the small, in the large. Computer should wait, not human. **#CI**
  - Almost anything can be made faster. Most often by 70%+ faster

Lean principles – inventory

- For CI to work in large environment lean thinking must be applied. Minimize the batch size. Remove waits and minimize work in progress
- Watch your commit & integration rhythm and keep it reasonable or know the good reasons why especially when working with multiple teams.
- Long interval between committing changes can mean you are building your software in too big chunks.
  - Large user stories, or refactorings. Too many unintegrated changes in your inventory might lead to integration problems especially with multiple teams.
- If your team members are not committing their changes in logical chunks (almost) every day you might have too much inventory building

Visibility in continuous integration

- CI system is a treasure trove of information
- Give just enough information on radiators. Make it understandable by anyone. Less is much more
- Make the information in radiators/histories understandable by anyone in project
- Mining the continuous integration database:
  - history - are builds breaking often during the sprint or two
  - are builds being done?
  - is software robust, same tests with no changes same results
  - are the tests finding the right problems - talk to people, compare tests
  - are the tests getting slower, do the tests break often – where
  - do components / teams add tests over time, do they fix tests quickly

Summary of the age of CI

- In larger projects a separate team may focus tooling development. **Persistent commitment is needed**.
- Have active sponsor/PO

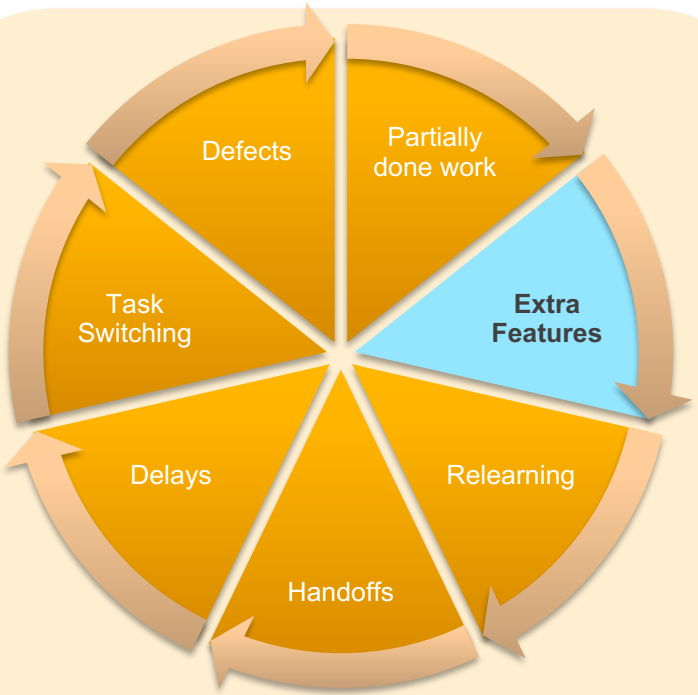
- CI works even in large projects.

- Sprint by sprint improvements – not big bang
- Good tests are important

- Context is important.

- Performance engineering will speed up the feedback – all teams need to participate.
- **Faster feedback time & visibility increases productivity**

Traditional wastes of development



Efficiency view

Addressed by Agile

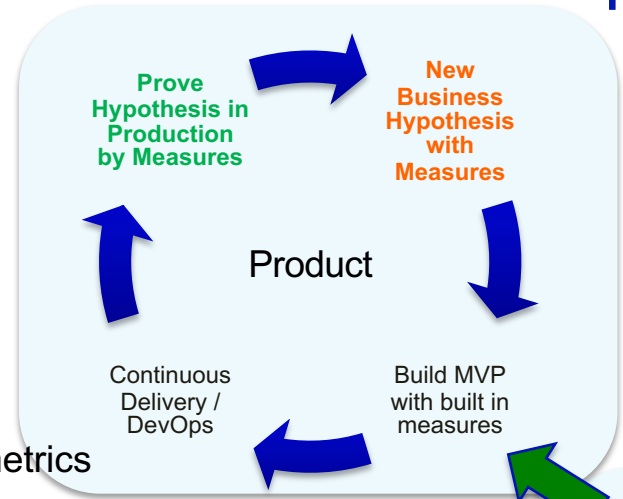
Biggest waste:

Building the wrong thing.

Effectiveness view

Addressed by principles of Lean Startup

Lean Startup - Effectiveness

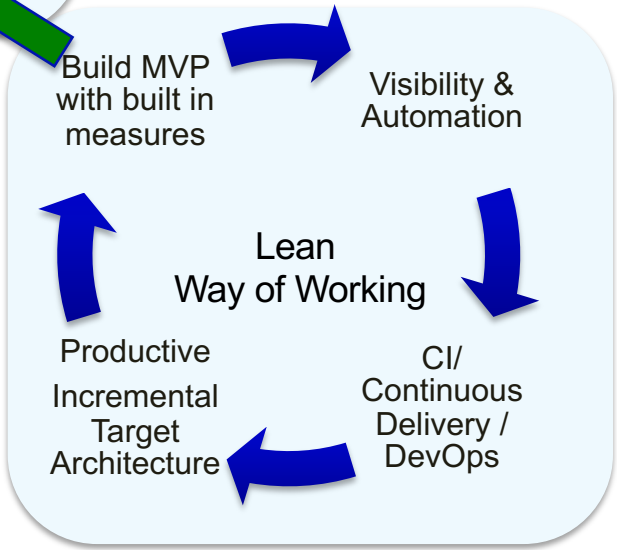


In R&D
- One week sprints

In Production
- Intra-day - One week – one month deliveries

Think output metrics
- Learning
- Delay
- Speed
- Example: Pirate metrics
  - Acquisition
  - Activation
  - Retention
  - Referrals
  - Revenue

Decrease time from hypothesis to proven business feature



Lean principles in agile – Efficiency

Reduce Inventory & WIP

Reduce Unimplemented backlog items

Reduce Bugs

Reduce age of bugs, code changes not in production

WIP Limit

- Unvalidated features
- Epics/ Features / stories in progress
- Size of stories
- Untested code
- Code changes not in production
- Compiled but not tested code

Improve Feedback time for

- User story invention to production
- Commit to
  - running the unit tests
  - running component functional tests
  - running system functional tests
  - running system performance tests
- Compile time
- Test time
- Installation time to
  - Test environment
  - Production environment

Eliminate Waste

- Missing testing -> unnecessary repeat loops
- Automate repeated manual steps (build, deployment)
- Automate manual tests
- Remove duplicated (repeated) work
- Reduce/remove interim products not used afterwards
- Remove unnecessary steps
- Eliminate feature flags when their time has gone

Maximize learning opportunities by having fast feedback loop

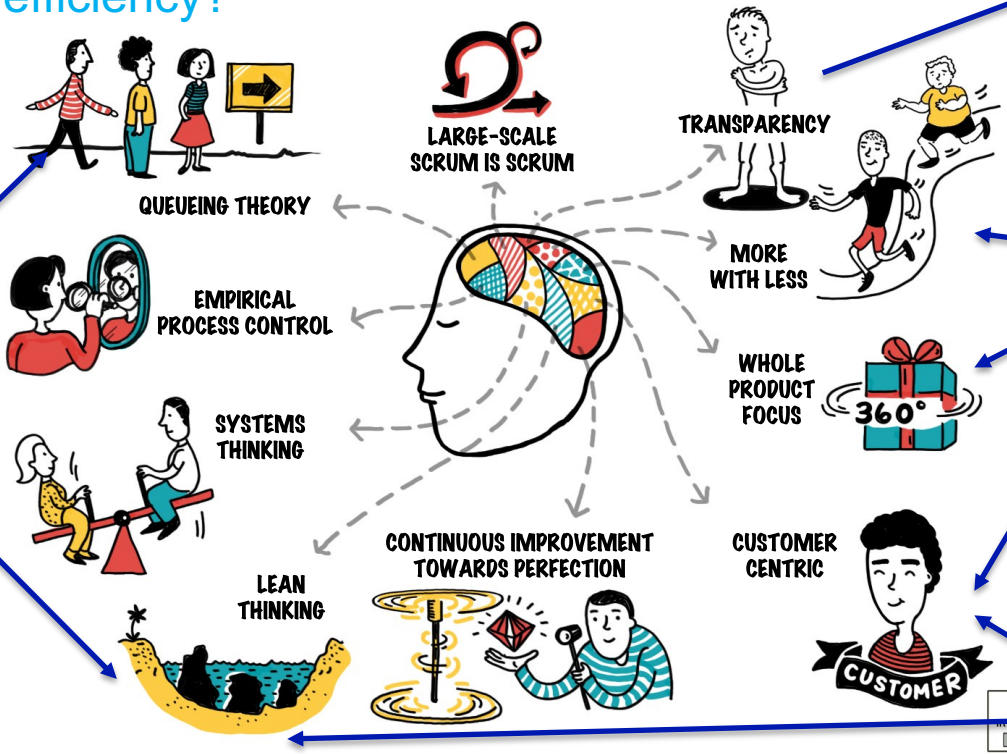
Prioritize biggest improvement potential that is solvable soonest.

Faster every sprint

Principles of scaling Agile (from Less)

Effectiveness or efficiency?

Visibility – Act on It



Fluent flow
Minimize inventory

MVP – MDP?

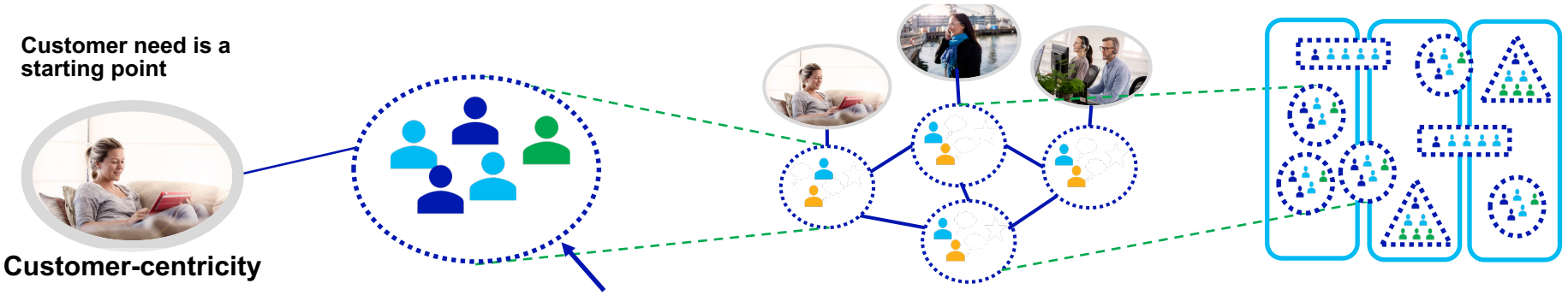
Lean Startup
"Get out of the Building"

Continuously Faster Feedback Loop – Increase Speed of Learning

Self organized team of teams **differ** from other forms of teams in several **distinct** ways

Rapid response with all the contextual information close to customer.

Alignment between teams concrete and rapid. Objectives built up from both directions



Decisions about how to respond in the customer facing team much more often

Management communicates clearly the purpose and direction

Transparent operative information available to all in team of teams

Information and work needs to flow easy and fast.
Agree on enough common things

Single team (ideal)
- BO, PO, Scrum team (end to end)
- same coffee machine
- same (open) space
- high frequency, high bandwidth communication

Short distance, few teams
- Kanban in the walls, system drawn in the walls, whiteboards, flipcharts
- Flowdock / Slack / MS Teams chat, Radiators

Long distance and/or many teams
- Shared easy tools
- Flowdock / Slack / MS Teams, Radiators
- Confluence, Jira boards, Gliffy

Common Ways of working
- Common – Concrete – Goals
- Common situational view (radiators)
- Definition of done
- Demo
- CI/CD/DevOPS
- -> Fast common builds, test & ops automation…
- Dependency handling…
  - Team chat, devgrooming, cross team 'gate' meetings
  - APIs, "Micro"services, Static analyzers

Scale by enabling the teams

- Aim for E2E self organised teams
  - No ”helper” / functional teams (eg ”testing”)
  - Grow **all the teams to be first class citizens** on their domain and eventually overall
  - Build and grow the structure of the team for the long term – consultants as specialty experts, advisors or work force – core or context?
- Only team of teams goals
  - for example: ways of working goals or concrete enough OKRs instead of individual goals
- Horizontal real time communications
- Minimize dependencies in tech and functions
- Common easy access to all the product artifacts and situational view

Example: Ways of working goals

Scale by improving ways of working

Don’t leave too much debt behind

- Low number of bugs, or get bitten at bad time
  - 1 bug per 3 developers open.
  - No old bugs
- Update the libraries – Whitesource or similar

Rapid iteration loops, constant delivery.

- Keep eye on the flow, small WIP
- CI, CD, DevOps do not call it ci if you just build stuff
- Minimize delivery chunks
- Coverage is safety net
- Automate…

Make everything visible and actionable

Horizontal real time communications

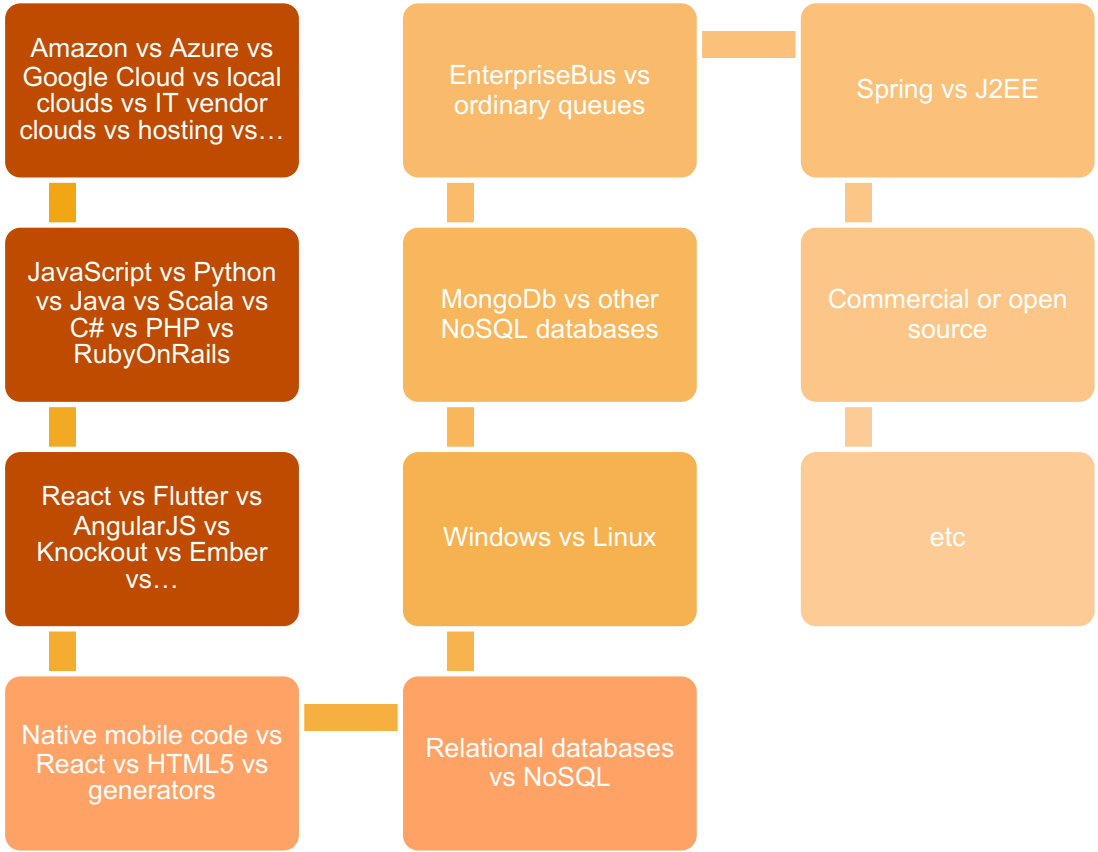
Involve regularly the cross team knowledge on common technical and functional teams

- Devgrooming, ”gates” discussion

Challenges in scaling

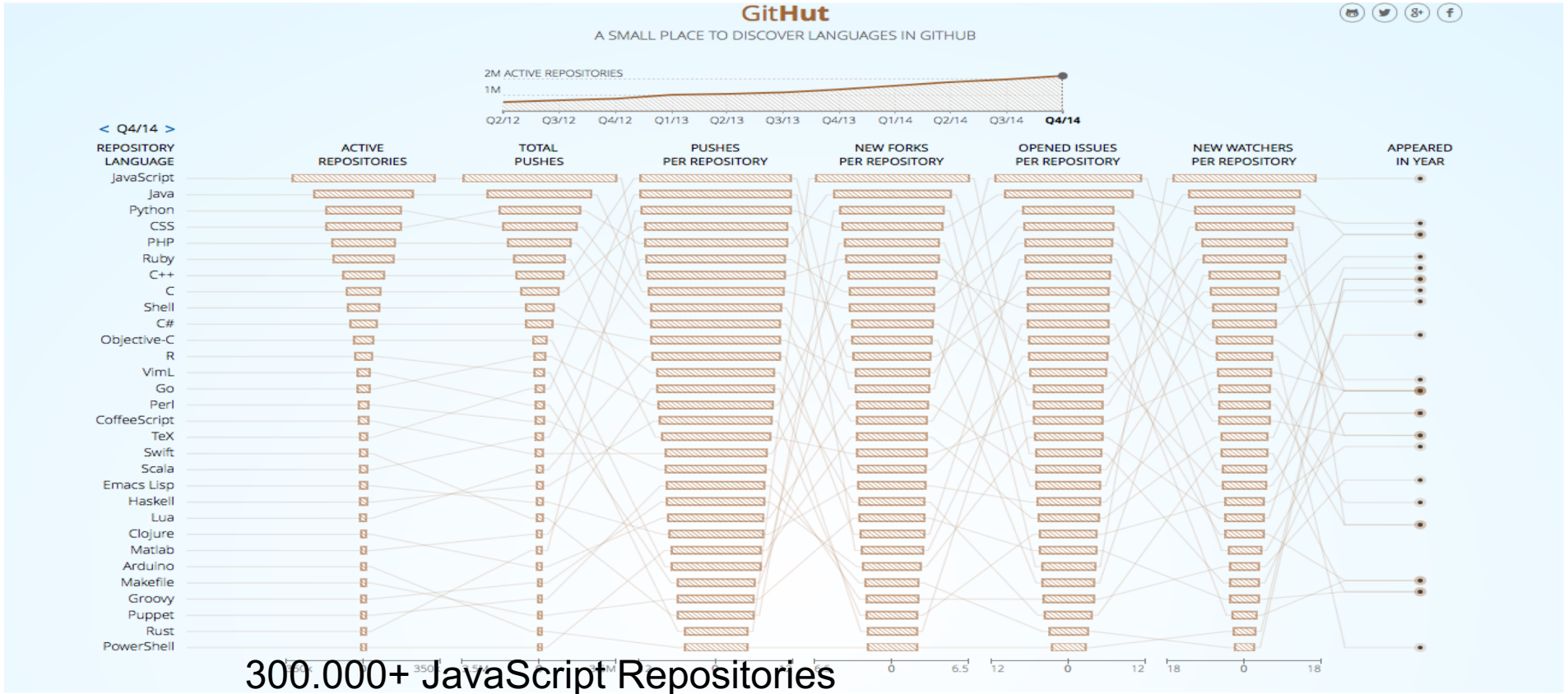
- Large timezone differences
- Siloes or silo objectives slowing down the flow
- Product dependencies : Functional, technical, external dependencies
- People dependencies: role, responsibilities, skillset (”Pete the Bottleneck”, ”HQ”)
- Cultural issues (team culture (”not willing to participate”, NIH, people culture)
- Large ”legacy” codebases
- Common ”design” activities across sites
- Maintaining the retrospective culture

World full of choices - opportunity or threat?



- …and this is only the technology stack part.

World full of choices - opportunity or threat?



300.000+ JavaScript Repositories

200.000+ Java Repositories

based on 2014 data

Hannu Kokko

“Don’t do your own
undifferentiated heavy lifting”
-Netflix

Don’t invent integration
use Enterprise Integration Patterns

http://www.enterpriseintegrationpatterns.com

Do not build without a really good reason…

- ✓ Queue (for example to a database)
- ✓ Workflow – engine
- ✓ Enterprise Integration Bus solution
- ✓ Full Javascript UI framework
- ✓ Own tx solution on top of NoSQL
- ✓ Reporting engine
- ✓ Persistence framework
- ✓ Rules engine
- ✓ Deployment framework
- ✓ …and there are other utility / framework categories…

Consider with care do you really want to

- Use commercial framework
- Build your own framework
- Take into use
  - consultant company's own framework
  - large open source FRAMEWORK
  - a new framework that does similar thing than an existing one
  - a second cloud
  - another database
  - ENTERPRISE version of anything

Take into use

- opensource library that is rare
- a generator
- commercial test framework
- framework with no upgrade path
- NoSQL database for product that needs lots of adhoc reporting

Have libraries, products, tools that are older than 1-2 years

Form "sufficient enough" approval method for components and frameworks to be taken into use.

Approve beforehand.

Follow ThoughtWorks Tech Radar

Think about lifetime of your system (applications and application portfolio)

- Ensure understanding of the solution in house from architects to production
  - Is your tech solution unique one – Why? Really – Why? Why…
  - Ensure availability of skills from several vendors / market
- Right amount of moving parts. For each purpose approximately one solution, at most two.
  - Beware “Flavor of the day”, Cool tech to consultant CV or real competitive advantage (productivity, time to market)
  - Security – more you have components, more you need to update
  - Think about skill needs, maintenance and support of your moving parts…
- Scalability: what means – plus 50% plus 100% plus 1000% increase to estimated usage
  - Machines, memory, cpu, disks, licenses

Predictability - Ways to keep your promises

- Keep sprints short
  - One week. Can be done to at least 100+ team size
- Definition of done – crisp and followed.
- Keep stories small – many per sprint.
  - 3+ per team per sprint at minimum. -> you will be able to deliver at least two of them…
  - ”testing”, ”technical task”, ”refactoring” etc is not user story
  - Do not play submarine – very very few big epics or stories.
  - Demo business level understandable stories once per week to your stakeholders
  - Honor your WIP limits
  - Get to the rhythm: Long term weekly average of promised vs delivered user stories 70% is a good number
- Keep bug count low and keep bugs young – less surprises and slowdowns
- Automate as you go
  - Relentlessly. You will not have time later.
- Deliver often.
  - Once a week – at least. Best in Finland are doing 5-10-20 a day – even in large teams

When you have to estimate: Capacity side

- Establish your predictability (teams delivering 70%+ of the committed user stories per sprint). Most stories should be sprint sized
  - Constant delivery -> no testing ”phase”.

- Capacity = How **many** user stories teams overall are capable of **delivering** per month / sprint over time.
  - Use the sum of teams more than individual teams number.
  - Monitor the numbers and do learning loop until it stabilizes.
  - Use the team level number to **advise** you when splitting the work to teams…
  - All the work included, not just developing the stories

- *Velocity must NEVER be incentivized KPI*
- *Do not compare teams velocity numbers*
- *No need to standardize user story size*

When you have to estimate: Demand side

- Those who know estimate roughly **the number** of stories and what they roughly (groups or names of stories) would be on whiteboard etc.
  - Split to reasonably sized Epics that can be delivered separately – MVP…
  - Record the estimated stories for the epic/s.
- Expansion factor
  - Monitor # of stories teams actually created in implementation = implementation stories
  - Sum of implementation stories / sum of estimated user stories (in a year or half a year) = **expansion factor**
  - Update the expansion factor as more stories gets built (every quarter / half a year)
- Demand = Number of estimated user stories x expansion factor
  - Is the Epic's number of estimation user stories x expansion factor reasonable for the business case. (as a fraction of capacity)
  - Roughly how/where does the Epic fit to **available capacity**…
- *Keep your epics small to deliver them early and to keep WIP low.*
  - *Ideally deliver every sprint or more often to production*
- *Team / sprint level estimation techniques are up to the teams – use any/many*

Possible portfolio views

- Business views (Innovation accounting, sales/costs) *#money*

- Capability view
  - Understand your capacity to deliver by quarter/month *#user stories*
  - What kinds of work the teams are doing *# user stories*
    - Architectural foundation
    - Refactoring in the large
    - Customer specific
    - Strategic initiative 1
    - Strategic initiative 2…
  - Codebase capability view
    - Which teams are changing what components
    - Knowledge, bottlenecks

Last words

- Prove business value quickly - fresh food business
  - **Fast to production** (weekly, daily or faster) – fast feedback & learning cycle – Lean startup/Agile/Kanban
  - **Common, interlocked goals** to all the teams
  - Cost is only one parameter – too easily measured. **Measure output, flow and effectiveness**
- **Long term commitment and support for improvement = leadership, management and culture**
  - **Small change every day** – many sprints of steady improvement can whittle down the largest issues
  - **Less Technical and Quality Debt = more flexibility**
  - Ways of working goals will make the teams better over time
  - Ensure skillset availability at your locations
- Do not build what you can get from open source or commercial
- **“Optimize” project portfolio over time** not single project

Thank You !