CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Continuous Integration, Delivery, Deployment, and DevOps

Agile Software Development

Eric Knauss
eric.knauss@cse.gu.se

Agenda today

- Continuous X:
  - Integration, Delivery, Deployment, and DevOps – Terminology and Origin
  - In the context of Large-Scale System Development
  - Continuous Deployment Practices
- Testing Strategies for Continuous X for Embedded Systems
- The Role of Value, Features, and Requirements

Key takeaways:

- Fundamental
  - Agile values
  - Testing
- Advanced
  - User value
  - System knowledge
  - Architecture

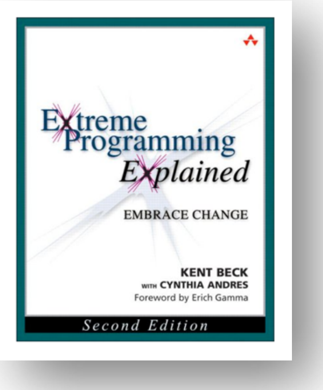
Part 1:

What is Continuous Integration (Delivery, Deployment) and why is it important

DEFINITION OF CONTINUOUS X

Continuous Integration: Key ideas

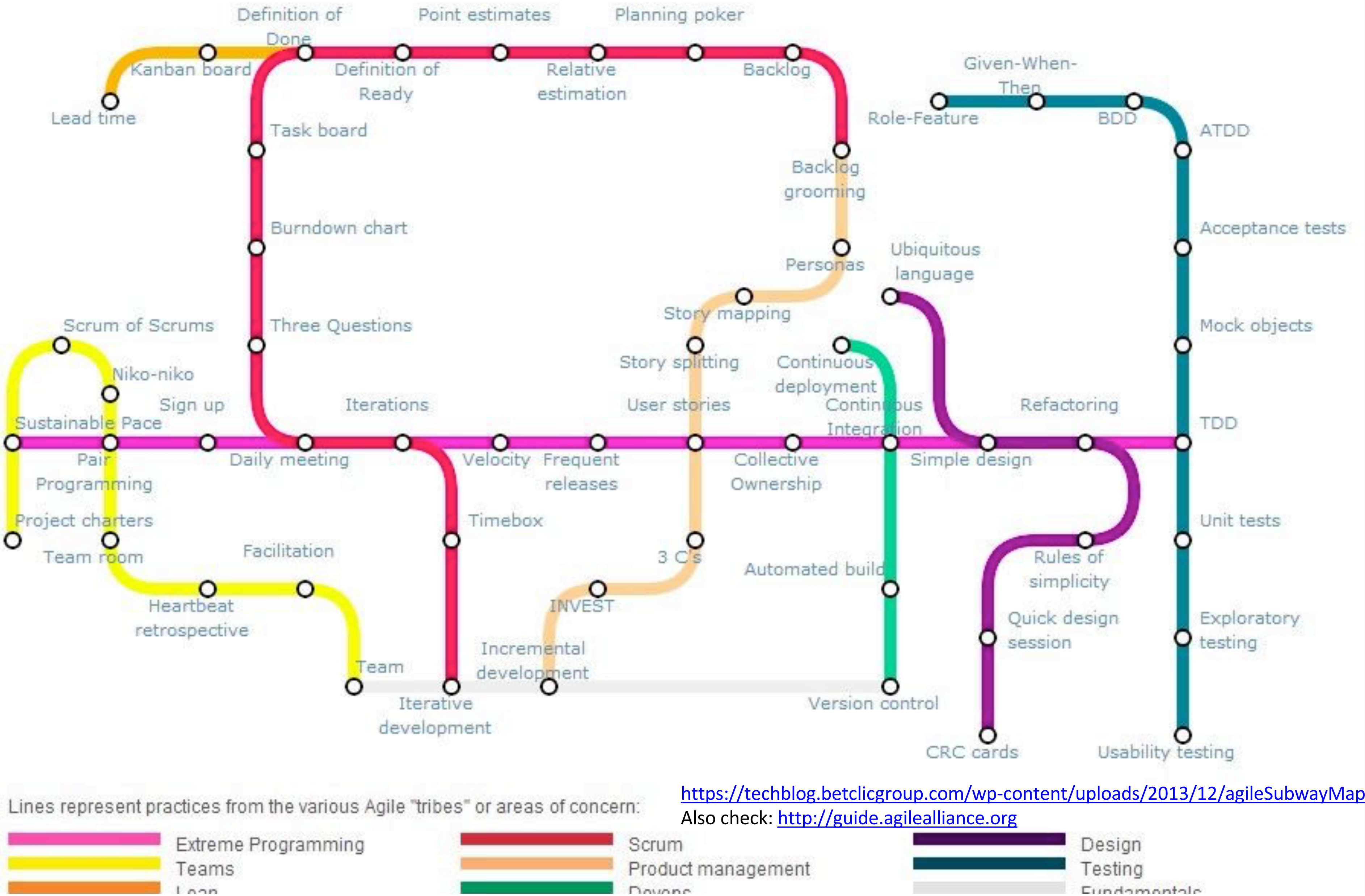
- Big Bang integration very costly
- Immediate feedback important
- Fowler: Mindset more important than technology



- Integrate and test code every few hours (1 day at the most)
- Dedicated machine helps
  - If Machine is free: pair sits down, integrates their changes, tests, and does not leave before 100% of tests run

Implies: small agile team

Origin: Agile Practice



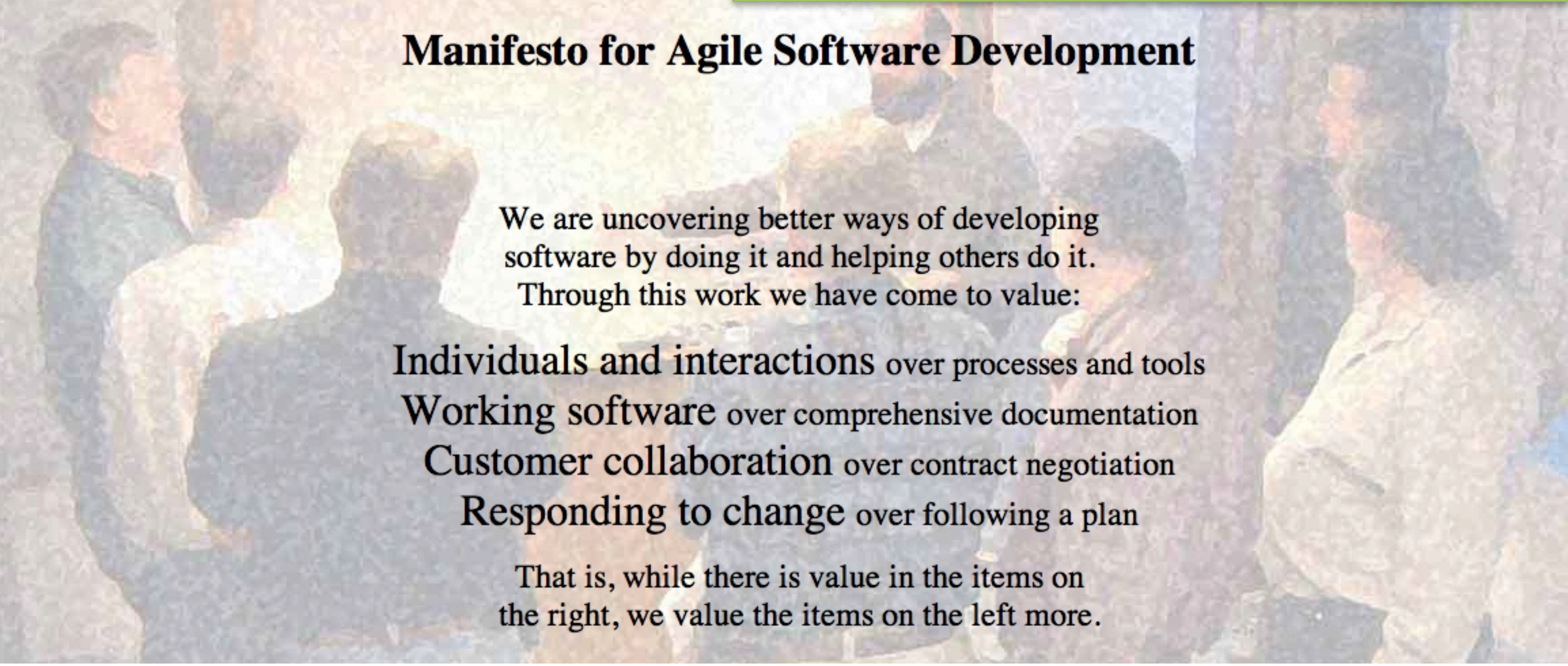
https://techblog.betclicgroup.com/wp-content/uploads/2013/12/agileSubwayMap

Also check: http://guide.agilealliance.org

Agile Manifesto

http://agilemanifesto.org

- Began as a provocation: Plan-driven development did not safe the Software world…
- Now a very serious movement, well adapted in industry.
- There are a couple of established agile methods: How to integrate these values in everyday software development



Manifesto for Agile Software Development

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.

What is agile? What is not?

- Agile – a compendium of ideas
  - Applied by number of methods (incl. XP, Scrum, Kanban, Lean Software Development)
- Core characteristics defined through
  - **Values**: General assumptions framing the agile view of the world
  - **Principles**: Core agile rules, organizational and technical
  - **Roles**: responsibilities and privileges of the various actors in an agile process
  - **Practices**: specific activities practiced by agile teams
  - **Artifacts**: tools, both virtual and material, that support the practices

[Mey2014]

Agile Values

1. Redefined roles for developers, managers, and customers
2. No ”Big Upfront” steps
3. Iterative development
4. Limited, negotiated functionality
5. Focus on quality, understood as achieved through testing

[Mey2014]

Agile Principles – Revised list

(according to [Mey2014])

Organizational

1. Put the customer at the center.
2. Let the team self-organize.
3. Work at a sustainable pace.
4. Develop minimal software:
   1. Produce minimal functionality.
   2. Produce only the product requested.
   3. Develop only code and tests.
5. Accept Change

6. Reflect regularly and improve continuously!

Technical

1. Develop iteratively:
   1. Produce frequent working iterations.
   2. Freeze requirements during iterations.
2. Treat tests as a key resource:
   1. Do not start any new development until all tests pass.
   2. Test first.
3. Express requirements through scenarios.

Why to do Continuous Integration

- Some evidence that
  - SW quality can be significantly improved
  - SW development can be significantly accelerated (time to market)
- Likely that
  - CI is not cheaper than traditional development
  - CI has positive impact on flexibility

→ Tendency to apply at scale and for systems

Organizational Categories

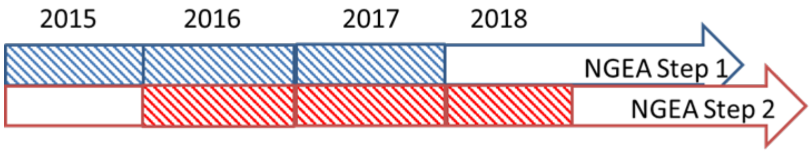
- **Enterprises**: software supports internal processes and external services (e.g. banks, insurance companies)
- **Product oriented organizations**: develop software or software intensive products that are not deployed directly by the development teams
- **Service companies**: develop software that is hosted and deployed directly by the developing team

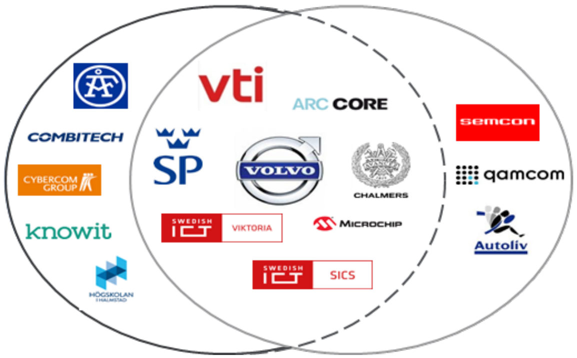
Matthew Bass (CMU)

Insights from Research Projects

Software Center aims to develop a ***strategic partnership*** with partner companies to ***significantly accelerate their adoption*** of novel approaches to software engineering

http://www.software-center.se/partners



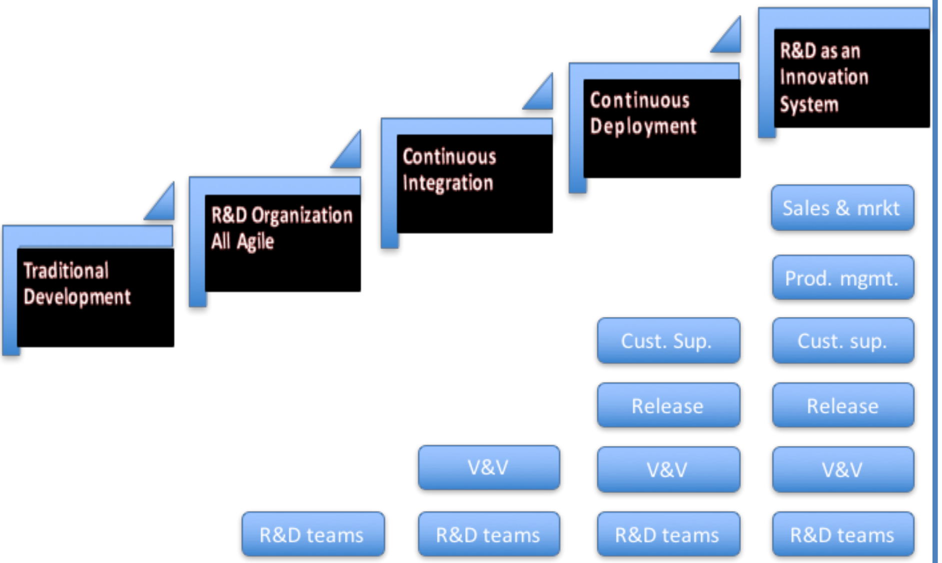


NGEA prepares the next generation of electrical architecture to increase flexibility, decrease development lead time, and increase the ability to develop complex systems and system of systems. These goals will be reached by consequently facilitating continuous integration, delivery, and deployment.

NGEA
Next Generation Electrical Architecture

STAIRWAY TO HEAVEN

The Stairway to Heaven Model describes the stages that companies evolve through when adopting novel approaches to software engineering



- Companies move through a predictable and repeatable pattern over time when evolving software engineering practices
- Each transition has business, architectural, process and organizational implications
- The higher up the stairway an organization climbs, the more organizational units are affected

- H.H. Olsson, H. Alahyari, J. Bosch, Climbing the"" Stairway to Heaven --A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software, 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA),, pp. 392-399, IEEE, 2012.

Context

- This is joint and ongoing work
  - Synergies between NGEA and Software Center Project 1 “Implications of Continuous Deployment”
  - Based on group interviews with 6 companies
    + cross-organizational workshops for consolidating results

- Research goal:
  - Many definitions, many concepts (Continuous Integration, Continuous Delivery, Continuous Deployment, …)
  - Understand which aspects are important in practice
  - Agree on shared language = *Common Understanding*

Agneta Nilsson

Magnus Ågren

Rogardt Heldal

Continuous Integration

NGEA

From Literature

Integrate new or changed code with the mainline codebase at frequent time intervals

From Interview

- Integrate / Test
- Sufficient quality of codebase
- One Mainline / Several
  - As long as each carry customer value
- Frequency Weeks / Days / Instant
- Why do we integrate – to get feedback

Synthesis

Integrate and test new or changed code with a codebase to enable feedback whenever we want

Continuous Deployment

The ability to release software whenever we want

- Release and Install in running / Deliver
- Higher quality level of codebase
- Strong quality assurance

Otherwise:
Continuous Delivery

The ability to release and install software in running system whenever we want

Working Definitions for this lecture

Definitions and Characterizations

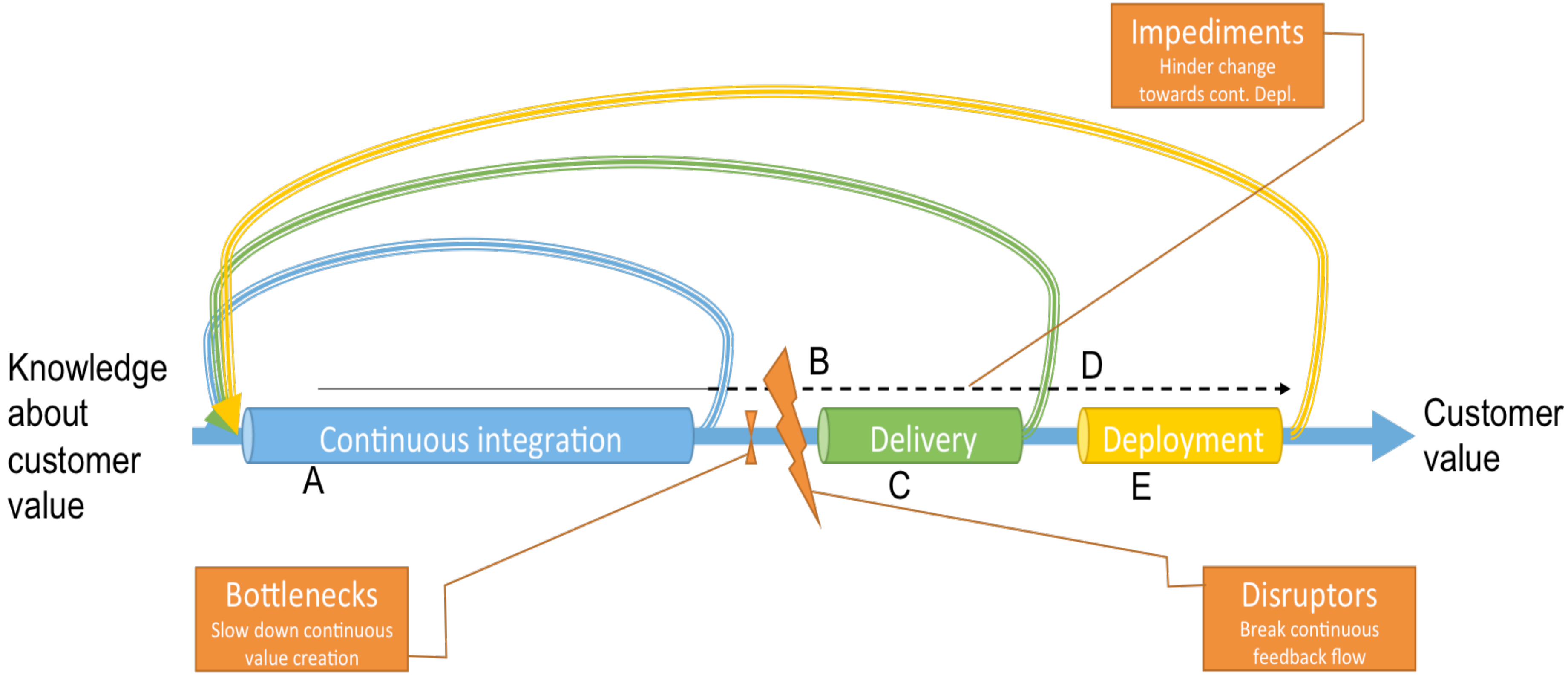
Continuous Integration: Ability to integrate and test new or changed code with a code base, which carries customer value, whenever desired to enable feedback.

Continuous Delivery: Ability to release customer and/or product value to a target environment whenever desired to enable feedback.

Continuous Deployment: Ability to release and install customer and/or product value into a running system at customer site whenever desired to enable feedback.

DevOps: Setup where cross-functional feature teams work closely with operations teams to facilitate continuous deployment

Some conceptual view


Impediments
Hinder change towards cont. Depl.
Knowledge about customer value
B
D
Continuous integration
Delivery
Deployment
Customer value
A
C
E
Bottlenecks
Slow down continuous value creation
Disruptors
Break continuous feedback flow

Something in between…

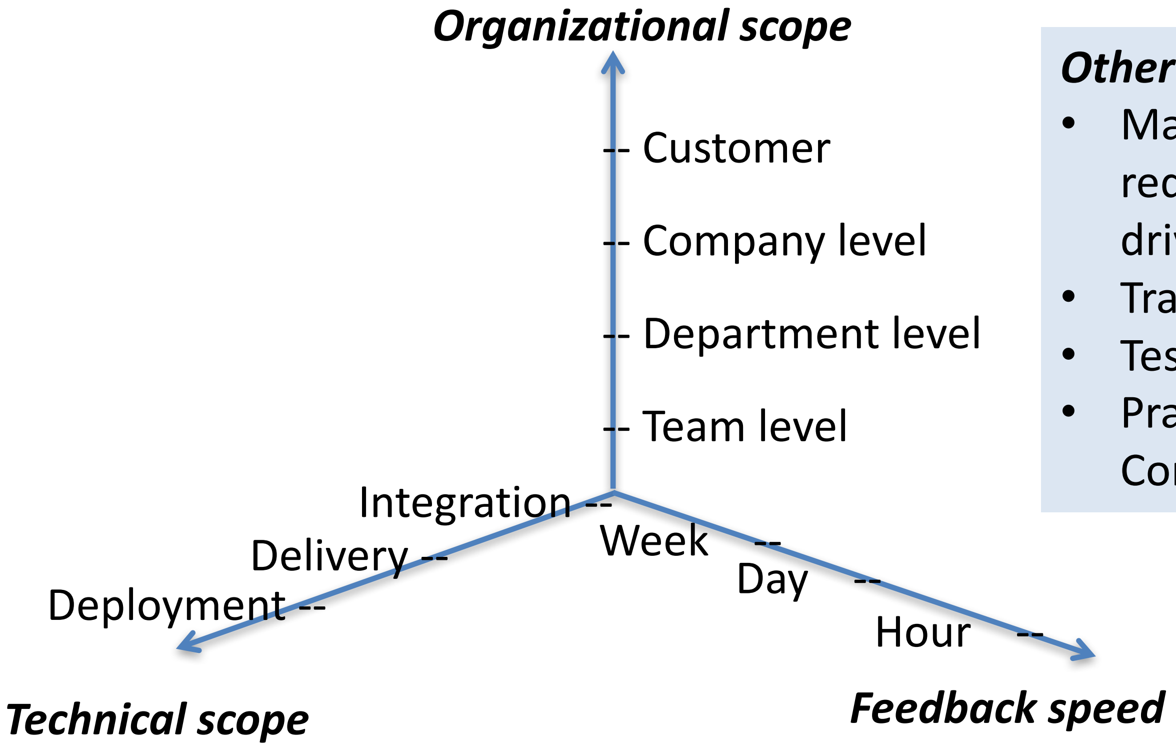
- Continuous Integration often on team level
- Continuous Deployment implies the whole organization to work in a continuous way

- Enterprise Continuous Integration [Ståhl and Bosch 2015]

- Continuous X includes a number of dimensions

Current work

- Systematic Literature Review to understand different dimensions of these terms



***Other dimensions*:**
- Managing architecture, requirements, business drivers
- Transparency
- Testing
- Practical scope: Vision, Constructive, Experience

Rahman, A. A. U.; Helms, E.; Williams, L.; Parnin, C.: Synthesizing continuous deployment practices used in software development. In: Proc. of Agile Conference; pg. 1-10; DOI: 10.1109/Agile.2015.12; Washington DC, USA. 2015

Continuous Deployment

Practice	Description
Automated deployment	Making software available to end-users automatically
Automated testing	Automated techniques to perform various testing activities (test case management, test monitor and control, test data generation, test case execution, …)
Code review	Requires developers to present software changes for comment and approval
Dark launching	Deploying software changes by keeping the functional aspects of the software changes hidden to end-users
End-user comm.	Communicating with end-users in order to receive feedback and gather requirements about the software of interest
Feature flag	(= feature toggle or feature flipper) is a technique that facilitates in triggering a specific branch amongst several branches of the software to enable or disable (parts of) features
Intercommunication	Sharing all necessary development and deployment information amongst software team members
Monitoring	Collecting deployment related information, producing appropriate performance metrics, and reporting them in an appropriate format
Software repository	Software library that contains all the necessary software artifacts. We must distinguish **branch and trunk shipment**.
Staging	Executing a specific set of techniques by the adoptee after software changes are written, tested, and before software changes are deployed to end-users
Dogfooding	When a software team uses its own software as part of their software development
Gradual rollout	Deploying software changes step-by-step to fractions of end-users

Part 2:

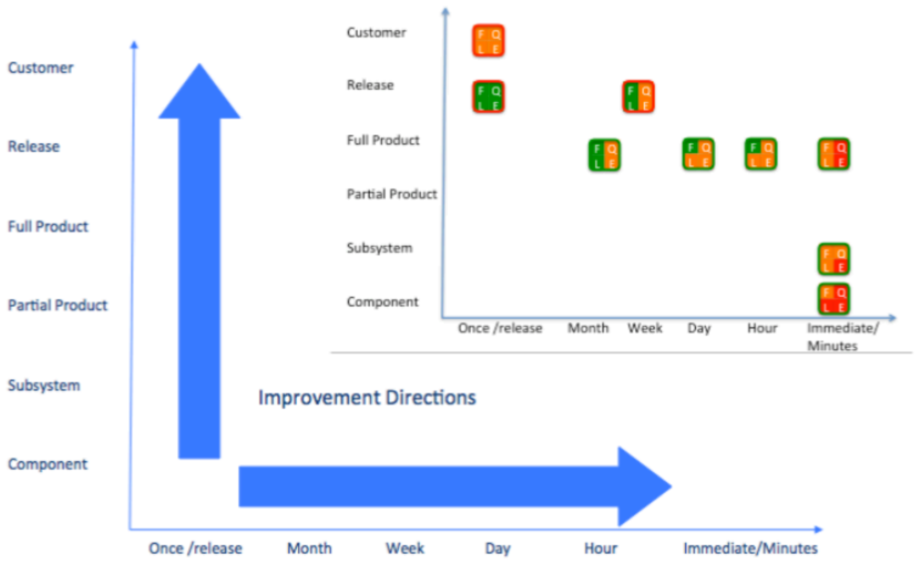
Automation and Dynamic Optimization of Test Suites

TESTING STRATEGIES FOR CONTINUOUS X FOR EMBEDDED SYSTEMS

CIVIT

Software Center

The CIVIT model is a test process improvement technique with the purpose to visualizing the end-to-end testing activities involved (from component to product level) to create a shared understanding of the current situation and support the identification of improvement areas.

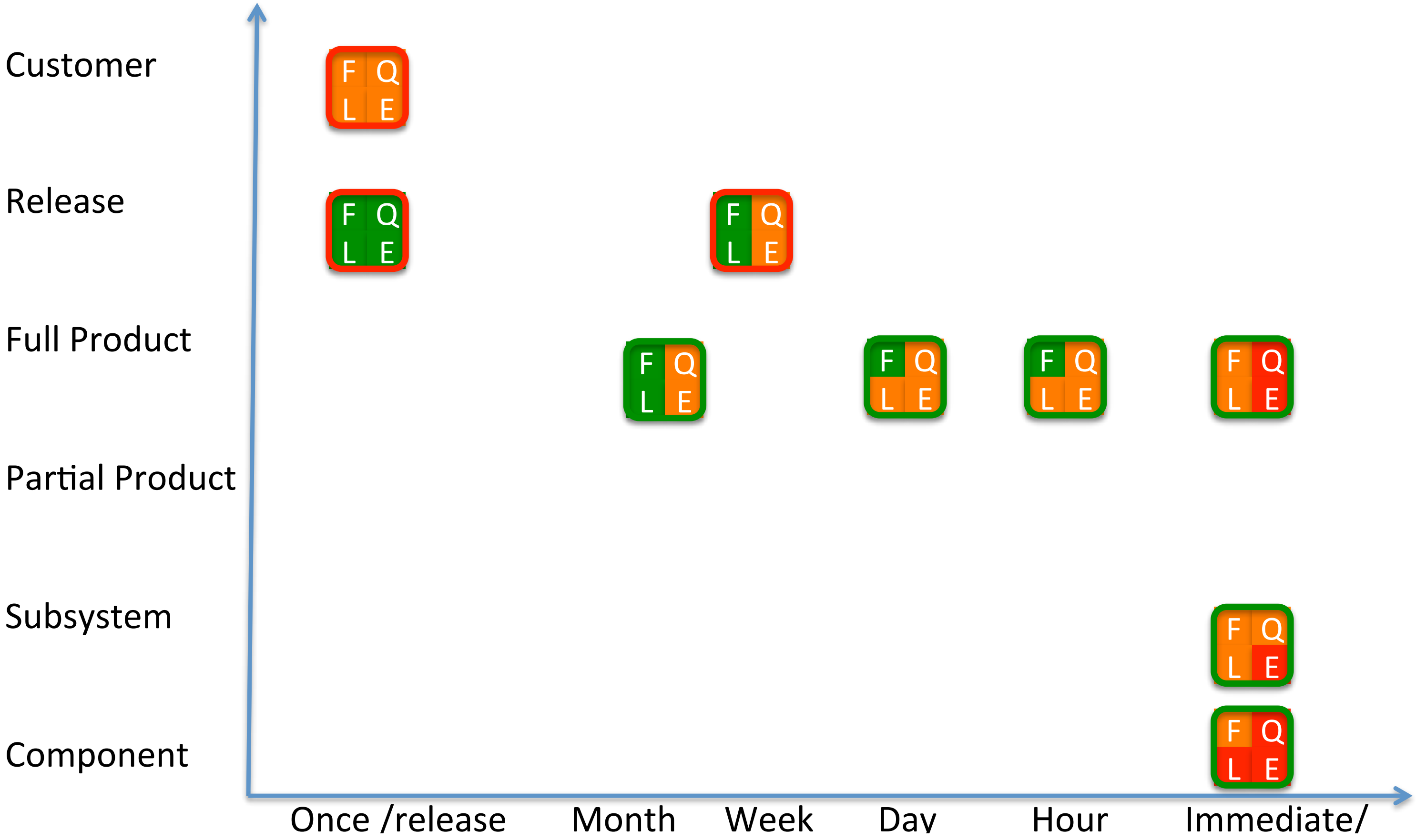


- Often a lack of an adequate overview
- Tend to lead to double work, slow feedback loops, issues found too late, disconnected organizations, unpredictable release schedules
- It enables a solid understanding of the end-to-end testing activities
- Particularly useful as a basis for discussion, to identify problems and to reason about suitable measures

- Nilsson, A., Bosch, J. and Berger, C. (2014) 'Visualizing testing activities to support continuous integration: A multiple case study'. In proceedings of Agile Software Development, XP, Rome, Italy, 26-30 May, 2014. Springer Volume 179, pp. 171–186

For more information please contact agneta.nilsson@gu.se

Example Civit model



Customer

Release

Full Product

Partial Product

Subsystem

Component

Once /release

Month

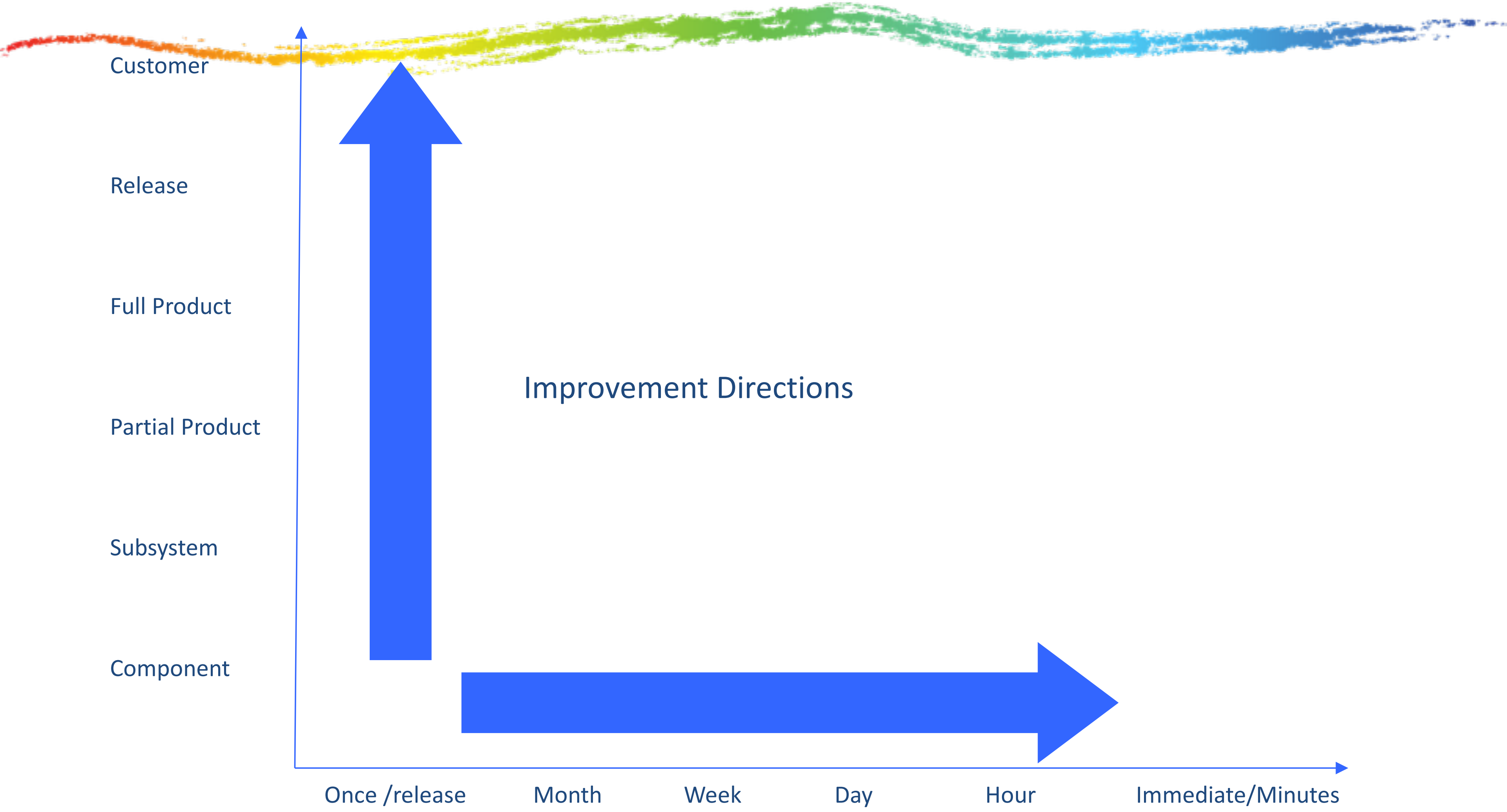
Week

Day

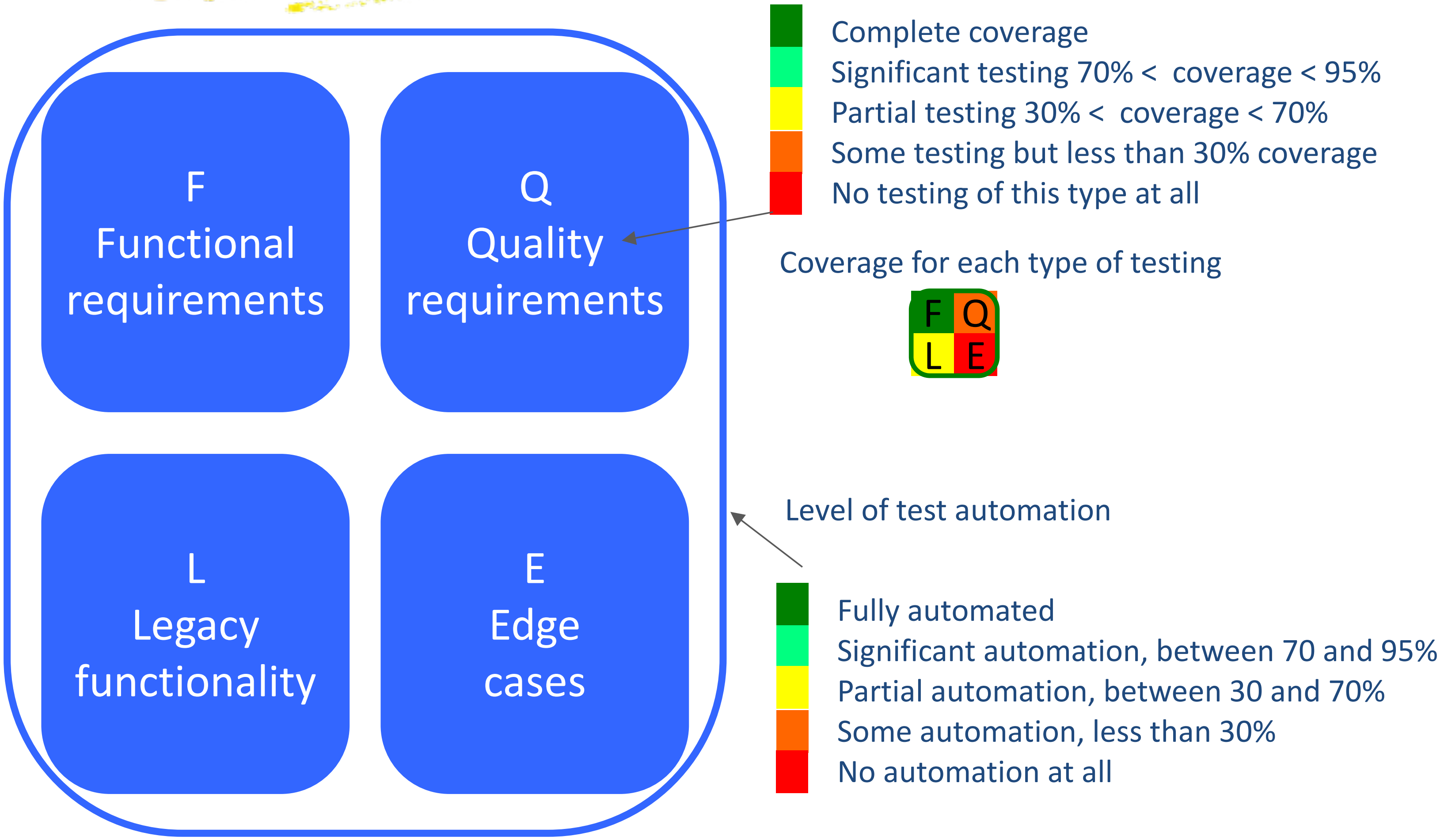
Hour

Immediate/

The CIVIT Model



Legend



F
Functional requirements

Q
Quality requirements

L
Legacy functionality

E
Edge cases

- 🟩 Complete coverage
- 🟩 Significant testing 70% < coverage < 95%
- 🟨 Partial testing 30% < coverage < 70%
- 🟧 Some testing but less than 30% coverage
- 🟥 No testing of this type at all

Coverage for each type of testing

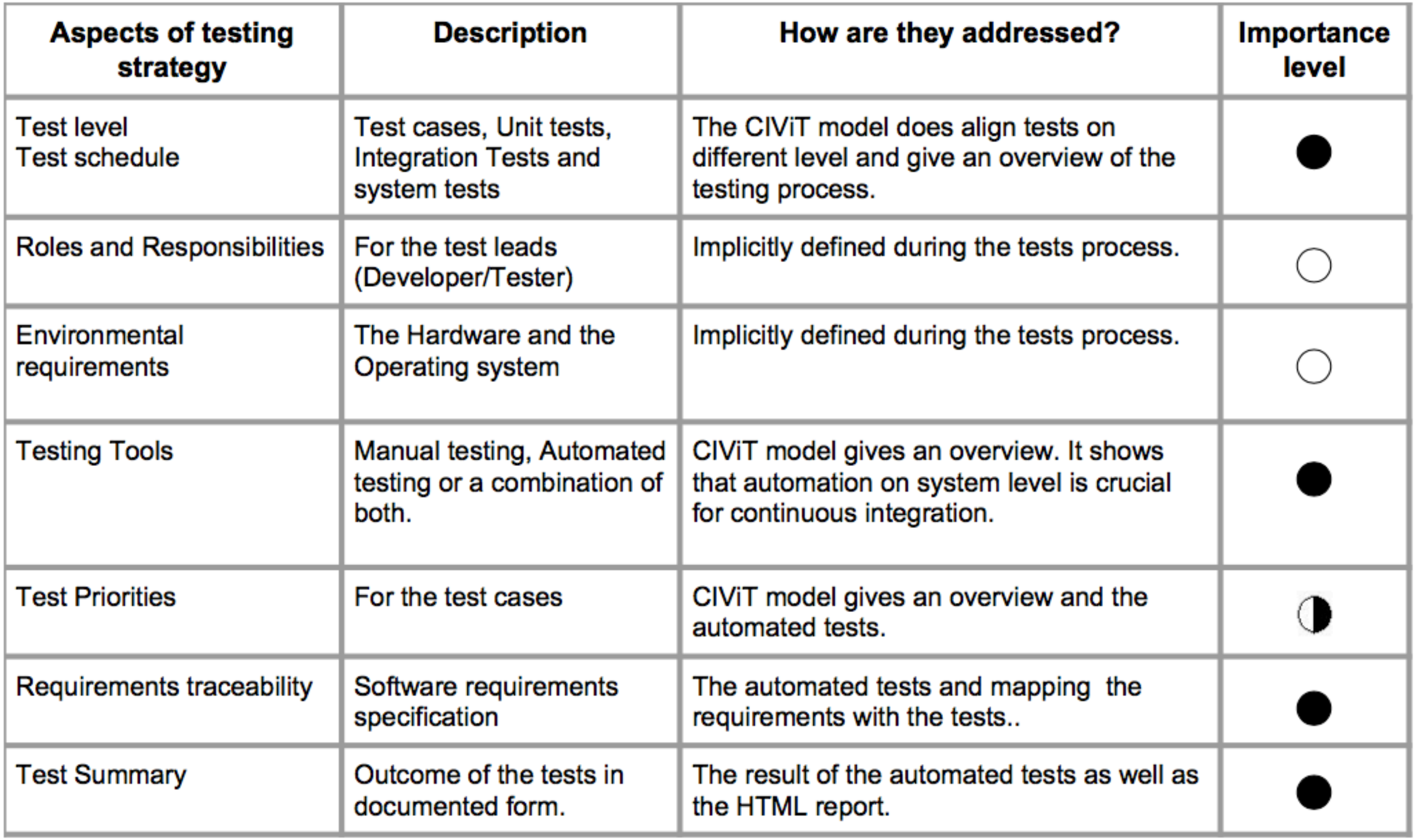
F	Q
L	E

Level of test automation

- 🟩 Fully automated
- 🟩 Significant automation, between 70 and 95%
- 🟨 Partial automation, between 30 and 70%
- 🟧 Some automation, less than 30%
- 🟥 No automation at all

Testing Strategies for CI

+ Test lifecycle

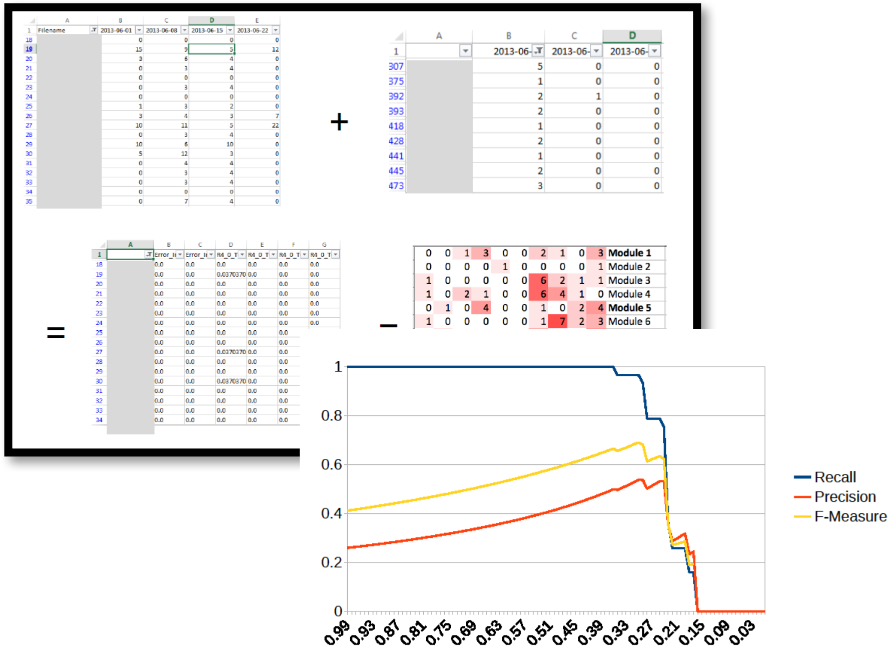


Aspects of testing strategy	Description	How are they addressed?	Importance level
Test level Test schedule	Test cases, Unit tests, Integration Tests and system tests	The CIViT model does align tests on different level and give an overview of the testing process.	●
Roles and Responsibilities	For the test leads (Developer/Tester)	Implicitly defined during the tests process.	○
Environmental requirements	The Hardware and the Operating system	Implicitly defined during the tests process.	○
Testing Tools	Manual testing, Automated testing or a combination of both.	CIViT model gives an overview. It shows that automation on system level is crucial for continuous integration.	●
Test Priorities	For the test cases	CIViT model gives an overview and the automated tests.	◑
Requirements traceability	Software requirements specification	The automated tests and mapping the requirements with the tests..	●
Test Summary	Outcome of the tests in documented form.	The result of the automated tests as well as the HTML report.	●

Haritha Gangineni Sarah Jamil: Testing Strategies to Support Continuous Integration for Complex Systems. Master's thesis in Software Engineering, Chalmers and University of Gothenburg, 2016

CCTS

The Code-Churn Test Selection model identifies the most optimal test suite based on the changes in the source code



	A	B	C	D	E
1	Filename	2013-06-01	2013-06-08	2013-06-15	2013-06-22
18		0	0	0	0
19		15	9	5	12
20		3	6	4	0
21		0	3	4	0
22		0	0	0	0
23		0	3	4	0
24		0	0	0	0
25		1	3	2	0
26		3	4	3	7
27		10	11	5	22
28		0	3	4	0
29		10	6	10	0
30		5	12	3	0
31		0	4	4	0
32		0	3	4	0
33		0	3	4	0
34		0	0	0	0
35		0	7	4	0

	A	B	C	D
1		2013-06-	2013-06-	2013-06-
307		5	0	0
375		1	0	0
392		2	1	0
393		2	0	0
418		1	0	0
428		2	0	0
441		1	0	0
445		2	0	0
473		3	0	0

0	0	1	3	0	0	2	1	0	3	**Module 1**
0	0	0	0	1	0	0	0	0	1	Module 2
1	0	0	0	0	0	6	2	1	1	Module 3
1	0	2	1	0	0	6	4	1	0	Module 4
0	1	0	4	0	0	1	0	2	4	**Module 5**
1	0	0	0	0	0	1	7	2	3	Module 6

	A	B	C	D	E	F	G
1		Error_I	Error_I	R4_0_T	R4_0_T	R4_0_T	R4_0_T
18		0.0	0.0	0.0	0.0	0.0	0.0
19		0.0	0.0	0.0370370	0.0	0.0	0.0
20		0.0	0.0	0.0	0.0	0.0	0.0
21		0.0	0.0	0.0	0.0	0.0	0.0
22		0.0	0.0	0.0	0.0	0.0	0.0
23		0.0	0.0	0.0	0.0	0.0	0.0
24		0.0	0.0	0.0	0.0	0.0	0.0
25		0.0	0.0	0.0	0.0	0.0	
26		0.0	0.0	0.0	0.0	0.0	
27		0.0	0.0	0.0370370	0.0	0.0	
28		0.0	0.0	0.0	0.0	0.0	
29		0.0	0.0	0.0	0.0	0.0	
30		0.0	0.0	0.0370370	0.0	0.0	
31		0.0	0.0	0.0	0.0	0.0	
32		0.0	0.0	0.0	0.0	0.0	
33		0.0	0.0	0.0	0.0	0.0	
34		0.0	0.0	0.0	0.0	0.0	

- Reduction of test suite by 73% without any loss of effectiveness
- Can speed up continuous integration and reduce cycle times
- Can be applied at all test levels

- E. Knauss, M. Staron, W. Meding, O. Söder, A. Nilsson, M. Castell, “Supporting Continuous Integration by Code-Churn Based Test Selection”, Proceedings of the 2nd International Workshop on Rapid and Continuous Software Engineering (RCoSE), ICSE 2015, Italy.

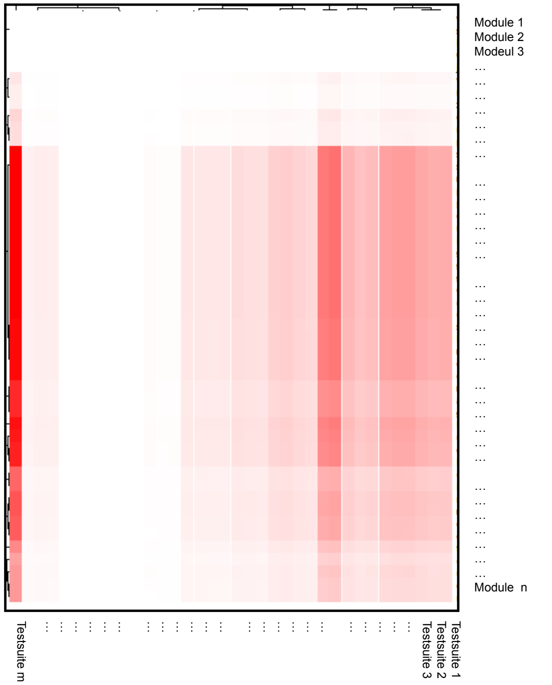
For more information please contact eric.knauss@cse.gu.se, agneta.nilsson@cse.gu.se and/or miroslaw.staron@cse.gu.se

Why test prioritization & selection?

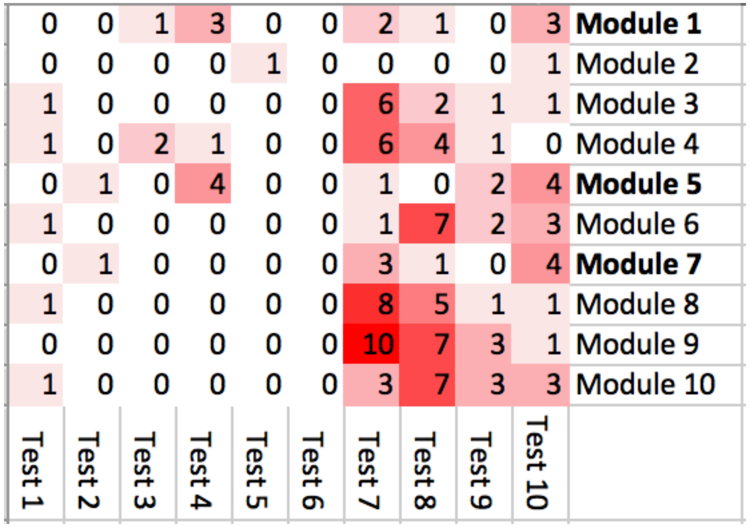
- Large product, many integration tests
  - Days or even weeks to run them all
- Continuous X = quick feedback
  - “We have 2h. Run the most important tests.”
  - “As a cross-functional team, I want to run important integration tests regularly so that integration will be smooth.”
  - “As an integration tester, I want to give feedback to developers as quickly as possibe.”

Recommending tests (idea)

- Prioritize tests based on heatmap
  - Collect which modules/components where changed
  - Sort tests by frequency that the failed in connection with collected changes
  - Cut off list by some criteria

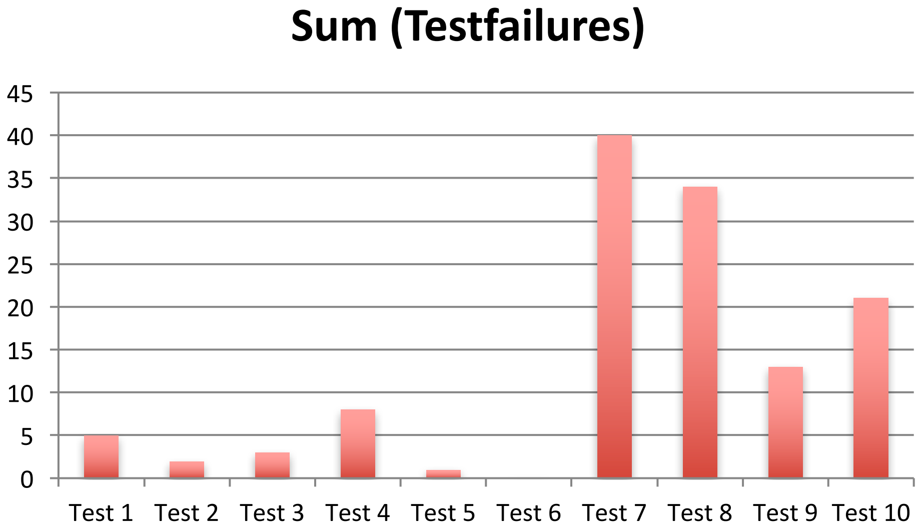


Principle 1



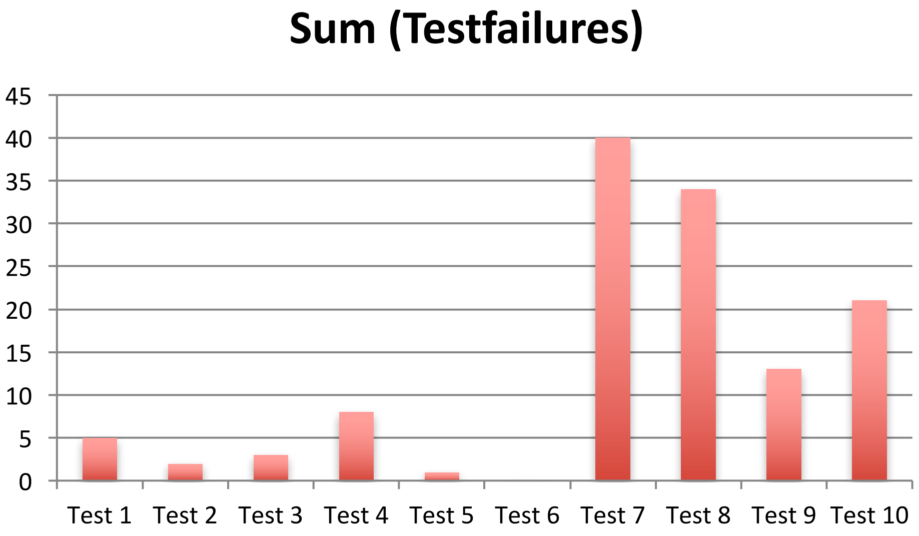
0	0	1	3	0	0	2	1	0	3	**Module 1**
0	0	0	0	1	0	0	0	0	1	Module 2
1	0	0	0	0	0	6	2	1	1	Module 3
1	0	2	1	0	0	6	4	1	0	Module 4
0	1	0	4	0	0	1	0	2	4	**Module 5**
1	0	0	0	0	0	1	7	2	3	Module 6
0	1	0	0	0	0	3	1	0	4	**Module 7**
1	0	0	0	0	0	8	5	1	1	Module 8
0	0	0	0	0	0	10	7	3	1	Module 9
1	0	0	0	0	0	3	7	3	3	Module 10
Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	

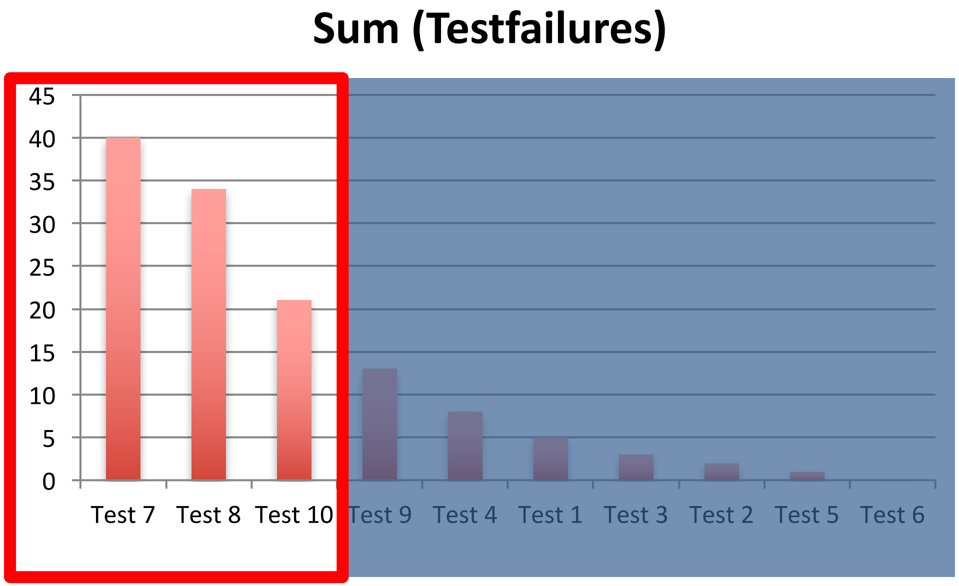
Starting from a heatmap…



…we can understand test efficiency

Principle

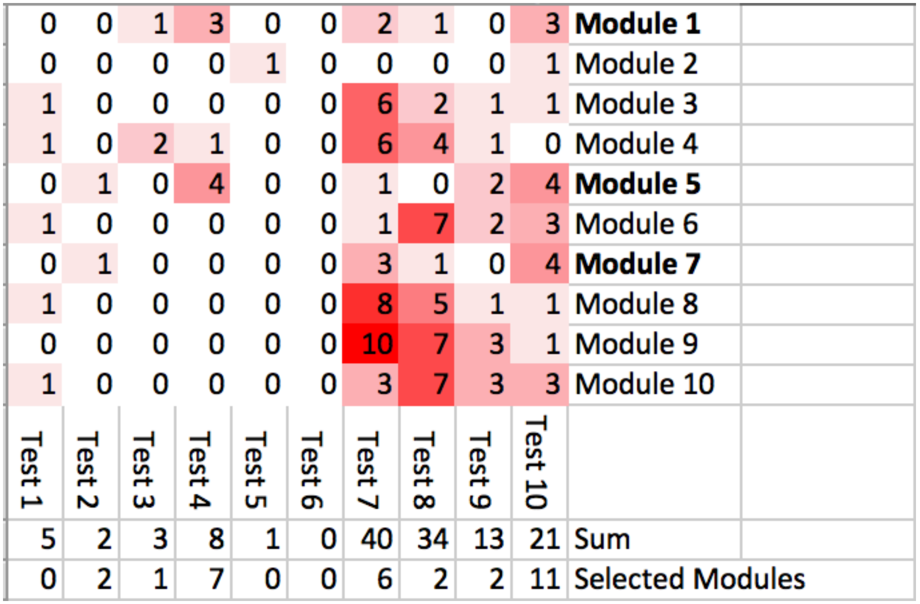




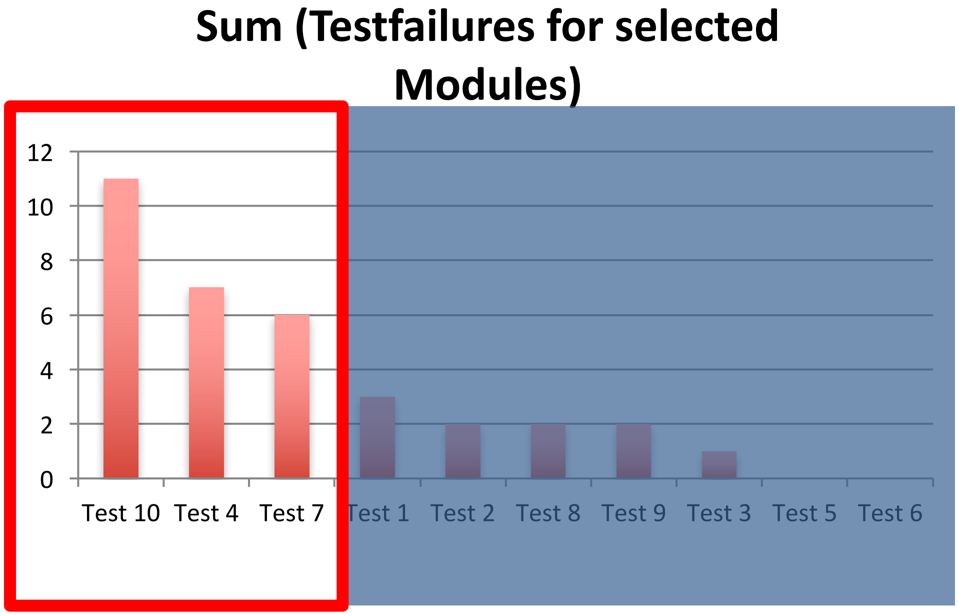
We can prioritize these tests (=sorting)

…and select tests.

Principle 2



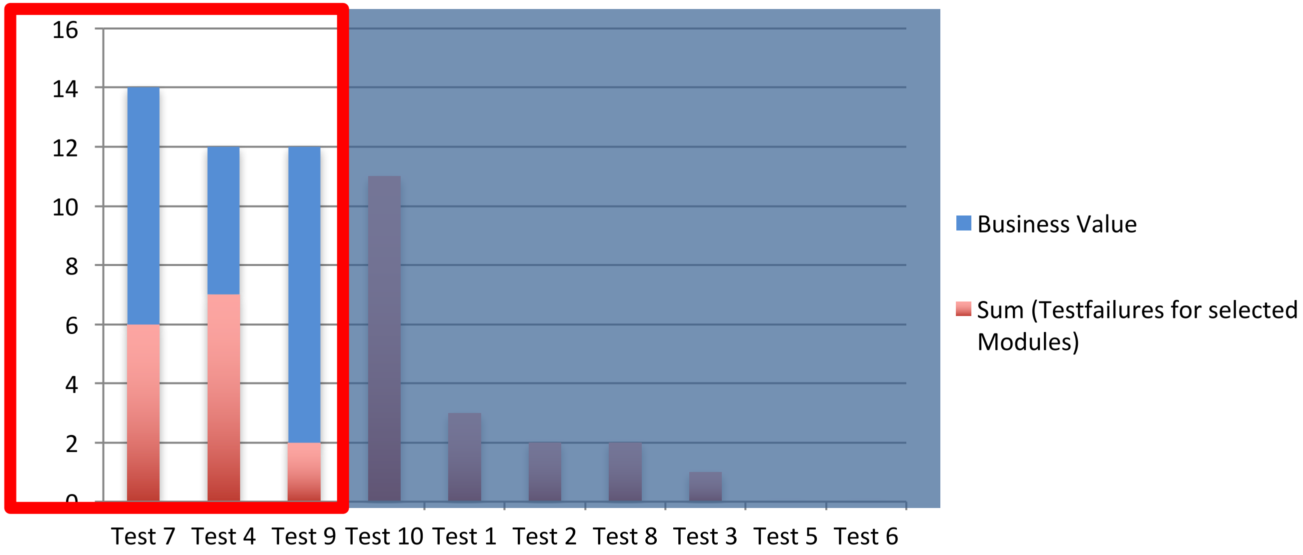
0	0	1	3	0	0	2	1	0	3	**Module 1**
0	0	0	0	1	0	0	0	0	1	Module 2
1	0	0	0	0	0	6	2	1	1	Module 3
1	0	2	1	0	0	6	4	1	0	Module 4
0	1	0	4	0	0	1	0	2	4	**Module 5**
1	0	0	0	0	0	1	7	2	3	Module 6
0	1	0	0	0	0	3	1	0	4	**Module 7**
1	0	0	0	0	0	8	5	1	1	Module 8
0	0	0	0	0	0	10	7	3	1	Module 9
1	0	0	0	0	0	3	7	3	3	Module 10
Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	
5	2	3	8	1	0	40	34	13	21	Sum
0	2	1	7	0	0	6	2	2	11	Selected Modules



If we know the modules that have recently changed…

…we can characterize test efficiency specifically for these modules

Principle 3



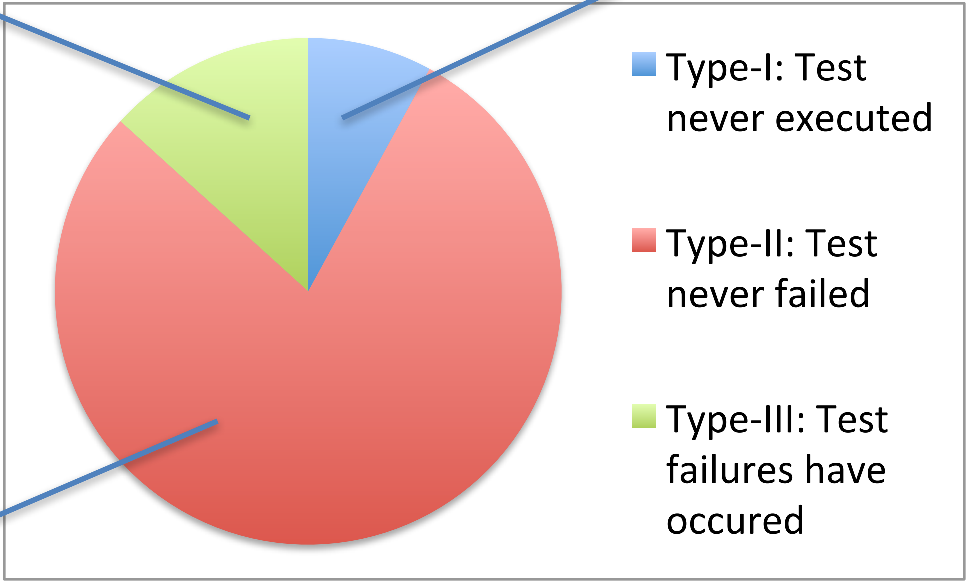
Perhaps a test is not likely to fail, but it would be strategically problematic. Therefore, we can also take into account business value

Applied to realistic data

Optimize the last 13%? But might be larger in your case!

Not a selection problem

Principle 2: Use historical data on code change



Linlin Wang: Implementation and Evaluation of an Automatic Recommender for Integration Test Cases. Master thesis at Chalmers, Gothenburg, Sweden, supervisor: Eric Knauss, examiner: Miroslaw Staron. 2015

Principle 1: Use historical data on test execution

918 Tests, history over one year (255 test executions)

Did not fail last year → don't run daily

Applied to realistic data (ongoing)

Principle 1: Use historical data on test execution

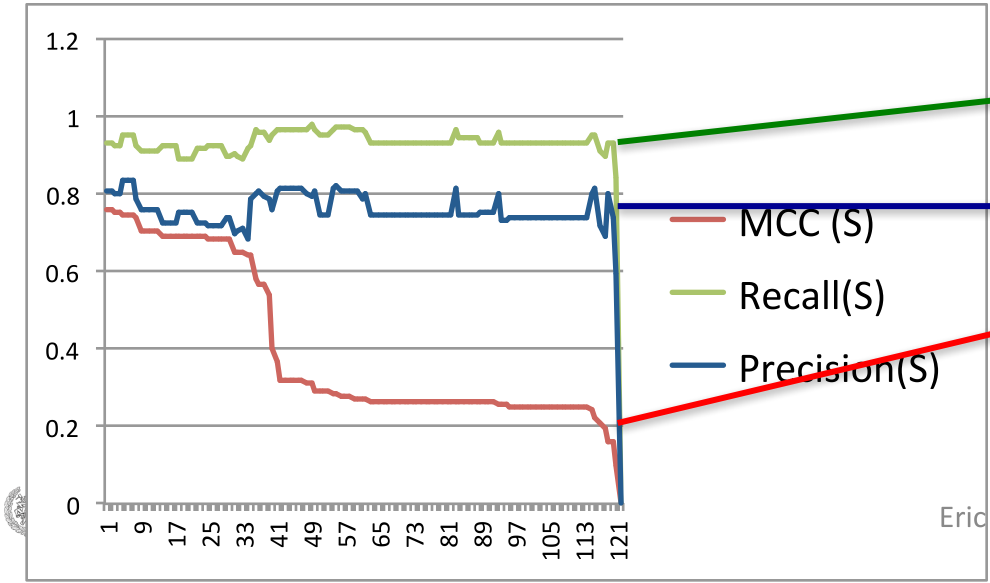
Did not fail last year → don't run daily

"You can do that tomorrow!"

Principle 2: Use historical data on code change

Optimize the last 13%? But might be larger in your case!

Still good potential for speedup.



If you apply Principle 2:

- *Only 6% of the tests that fail were not recommended*
- *Only 23% of the tests we did recommend don't fail*
- *Overall: **Moderate** strength recommendations (MCC > 0.2*
- *On average/for on third of the tests: **Strong** (MCC > 0.4) recommendations*

Part 3:

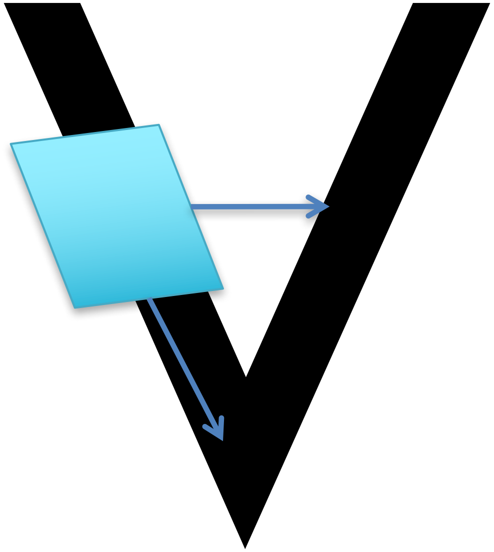
How to support flexibility, facilitate feedback, and maximize learning

THE ROLE OF VALUE, FEATURES, AND REQUIREMENTS

= "Stakeholder needs/properties a system should fulfill/exhibit"

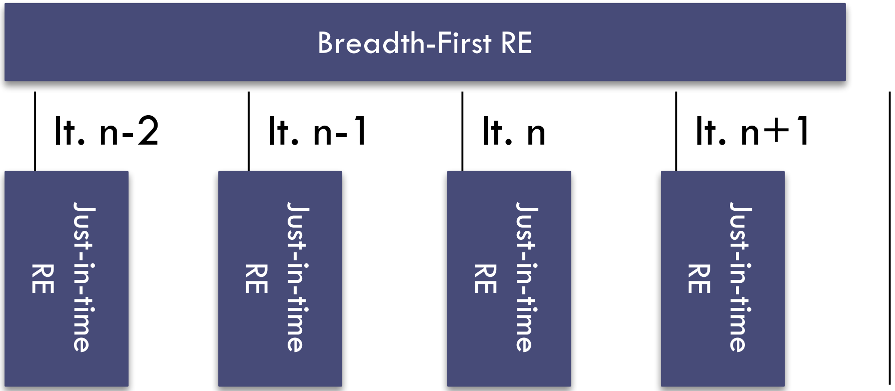
Requirements now and then

Then



Requirements a "waterfall phase" with specialists

Now



Breadth-First RE

It. n-2

It. n-1

It. n

It. n+1

Just-in-time RE

Just-in-time RE

Just-in-time RE

Just-in-time RE

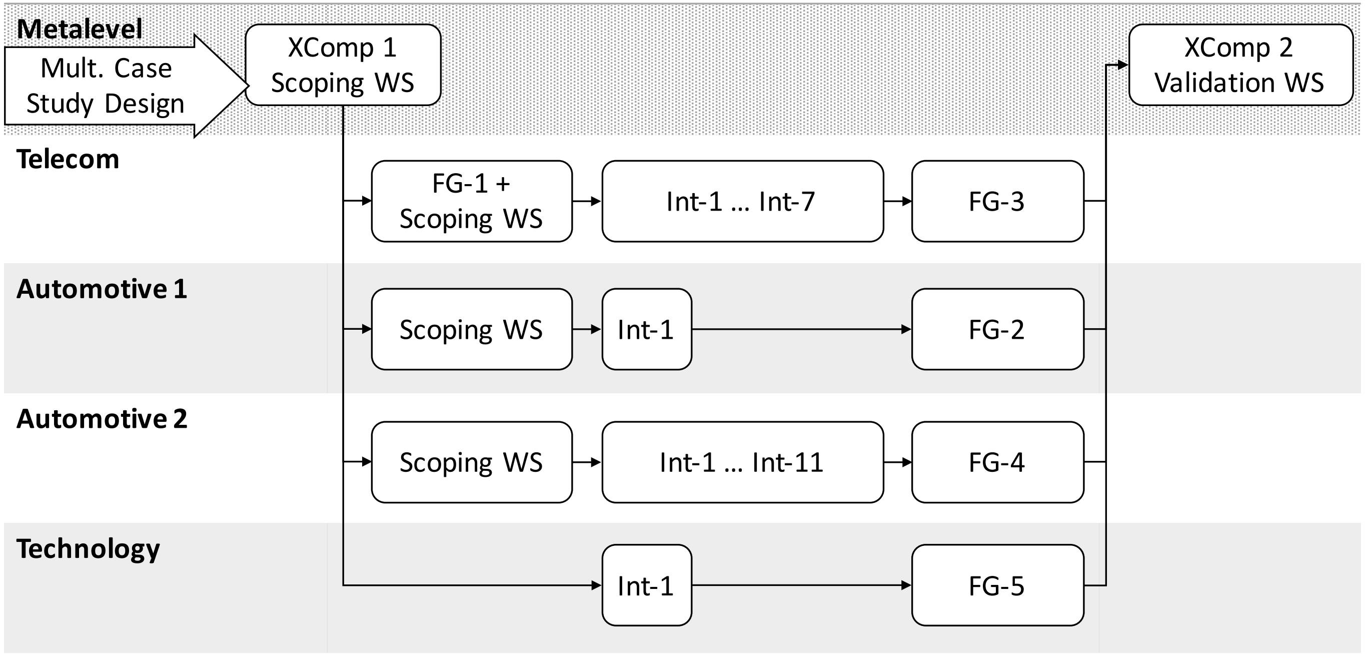
Requirements are everybody's responsibility

→ Knowledge Management Problem

Software Center

Mind the gap: Challenges with RE in Large-Scale Agile Systems Development

Case | **Data Collection** | **Discussion and Validation of results**



Metalevel

Mult. Case Study Design

XComp 1 Scoping WS

XComp 2 Validation WS

Telecom

FG-1 + Scoping WS

Int-1 … Int-7

FG-3

Automotive 1

Scoping WS

Int-1

FG-2

Automotive 2

Scoping WS

Int-1 … Int-11

FG-4

Technology

Int-1

FG-5

Comment

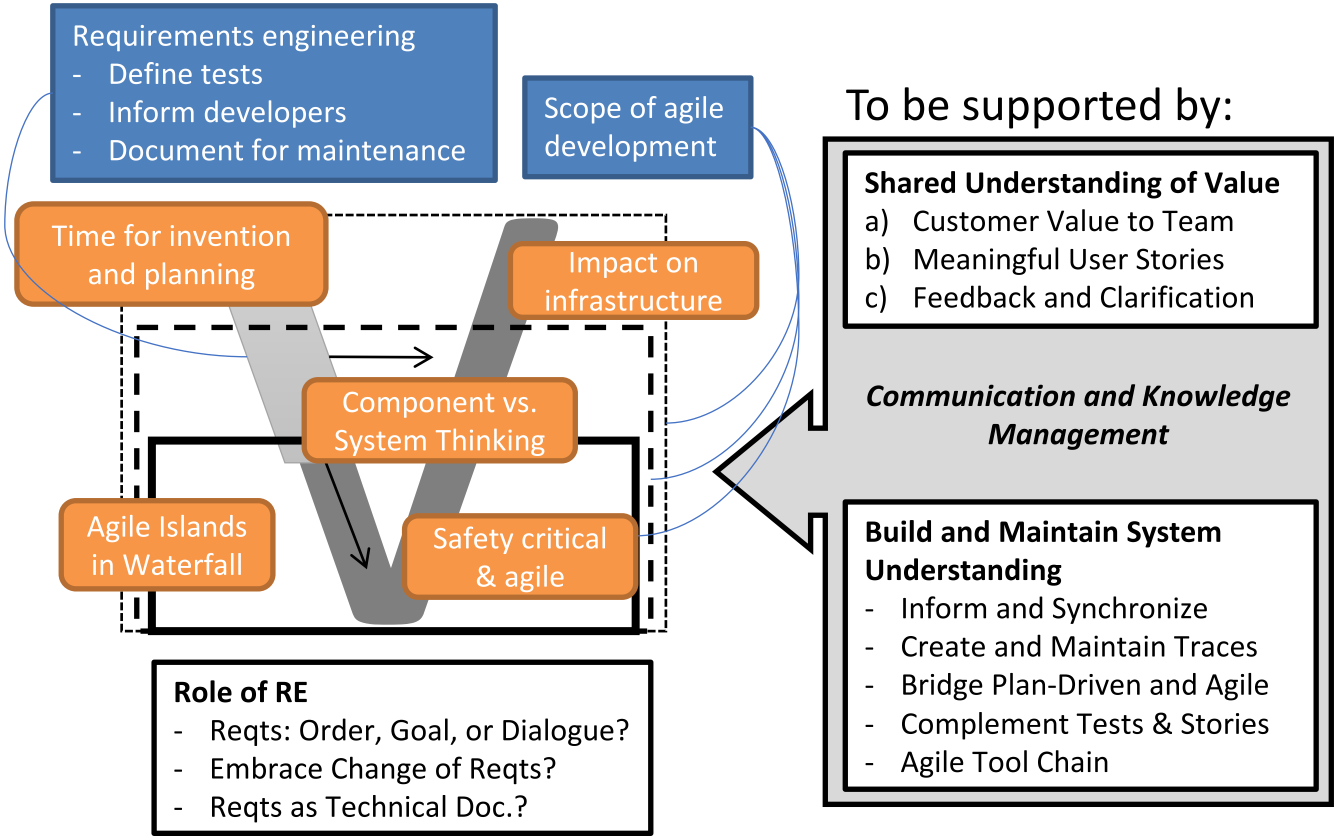
- Scope study
- Identify RQs

- Individual scoping and sampling for each case
- Selection of appropriate data collection method for each case

- Presentation of prel. Results

Kasauli, R.; Liebel, G.; Knauss, E.; Gopakumar, S.; Kanagwa, B.: Requirements Engineering Challenges in Large-Scale Agile System Development. Submitted to RE conference, 2017

Mind the gap:
Challenges with RE in Large-Scale Agile Systems Development



Requirements engineering
- Define tests
- Inform developers
- Document for maintenance

Scope of agile development

Time for invention and planning

Impact on infrastructure

Component vs. System Thinking

Agile Islands in Waterfall

Safety critical & agile

Role of RE
- Reqts: Order, Goal, or Dialogue?
- Embrace Change of Reqts?
- Reqts as Technical Doc.?

To be supported by:

Shared Understanding of Value
a) Customer Value to Team
b) Meaningful User Stories
c) Feedback and Clarification

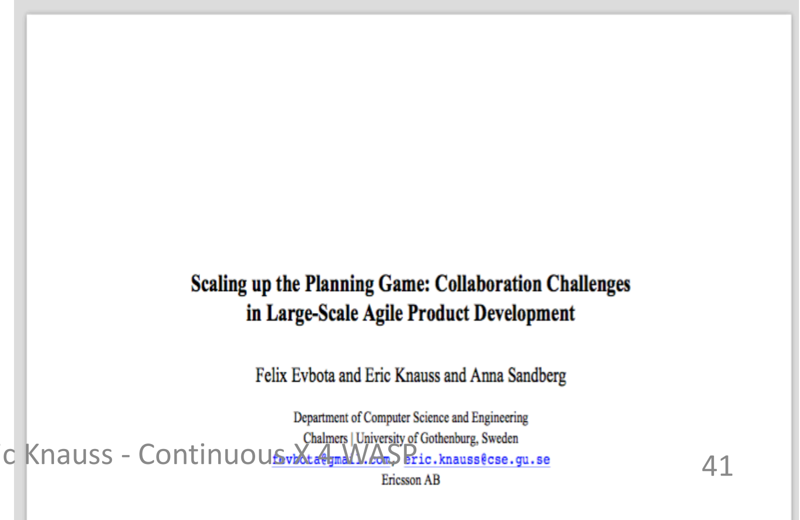
Communication and Knowledge Management

Build and Maintain System Understanding
- Inform and Synchronize
- Create and Maintain Traces
- Bridge Plan-Driven and Agile
- Complement Tests & Stories
- Agile Tool Chain

Kasauli, R.; Liebel, G.; Knauss, E.; Gopakumar, S.; Kanagwa, B.: Requirements Engineering Challenges in Large-Scale Agile System Development. Submitted to RE conference, 2017

LARGE-SCALE PLANNING GAME

Felix Evbota, Eric Knauss, Anna Sandberg: Scaling up the Planning Game: Collaboration Challenges in Large Scale Agile Product Development. In: Proc. of 17th Int. Conf. on Agile Software Development (XP 2016), Edinburgh, UK, 2016



Scaling up the Planning Game: Collaboration Challenges in Large-Scale Agile Product Development

Felix Evbota and Eric Knauss and Anna Sandberg

Department of Computer Science and Engineering
Chalmers | University of Gothenburg, Sweden
fevbota@gmail.com, eric.knauss@cse.gu.se
Ericsson AB

Understanding Customers and End-users

Large-Scale agile

- Huge distance to customers and end-users
- Hierarchy of product owners
- Coordinate efforts and dependencies of many teams
- Information silos vs. overload

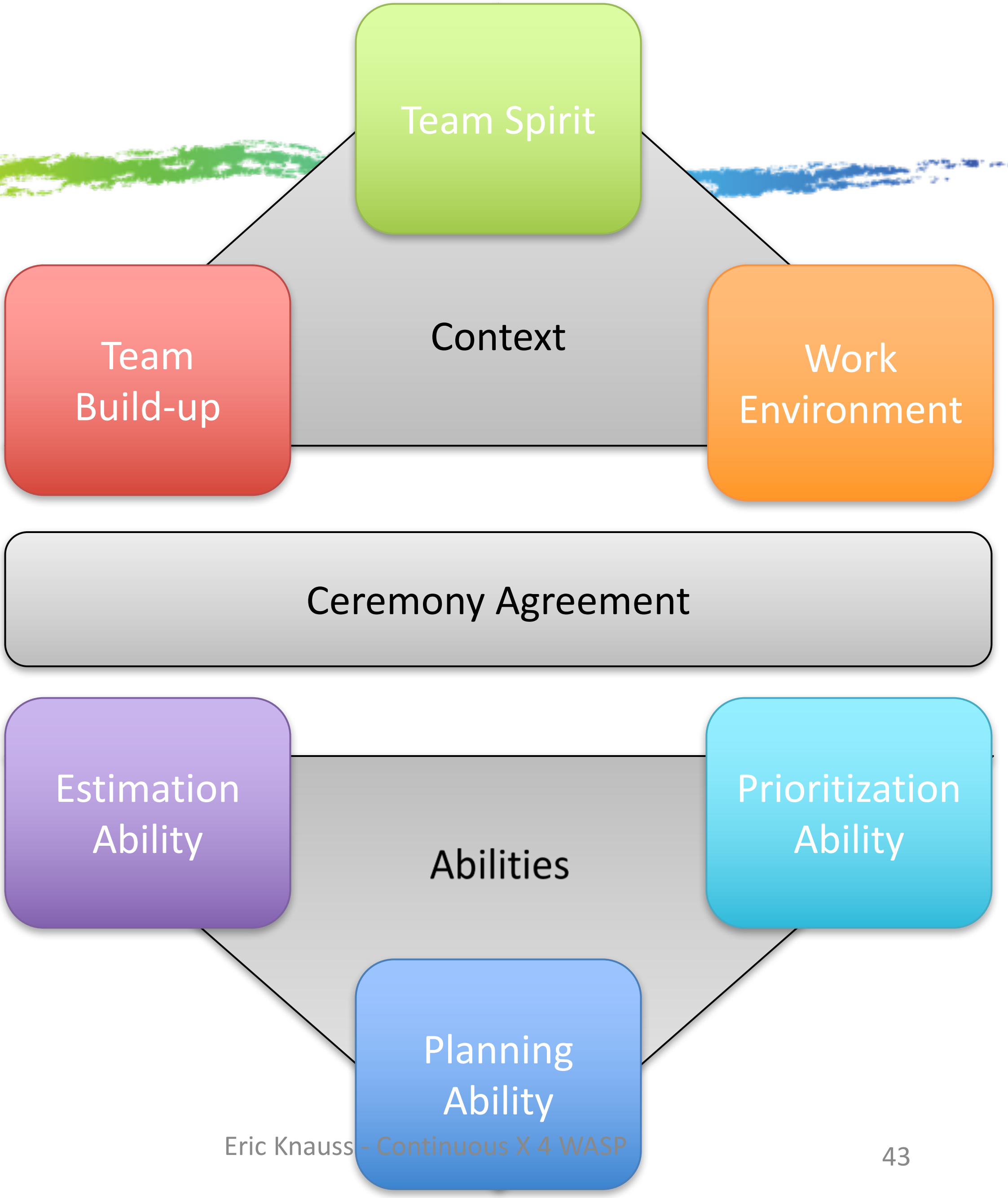
Agile

- Onsite customer / product owner
- User stories facilitate discussions
- Planning game to facilitate dialogue between customer and supplier
- Cross-functional team offers rich perspective

Findings

Research Method

- Qualitative Case Study (Ericsson)
- 10 Semi-Structured interviews with
  - op. product owner,
  - line manager,
  - program leader,
  - project leader,
  - release leader,
  - team leader and
  - developer

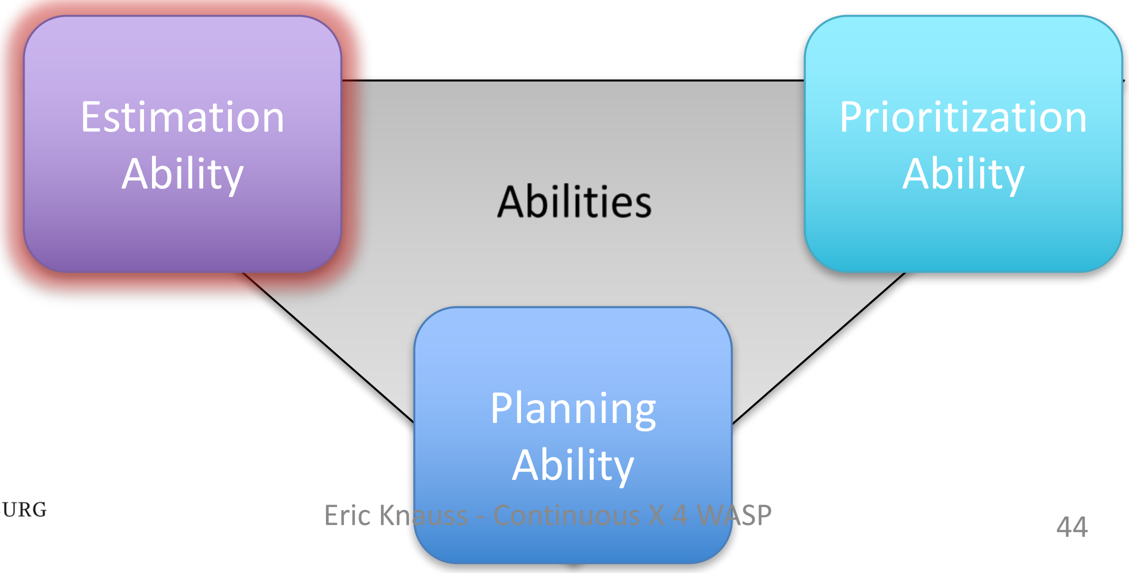


Findings

"[Previously we] estimated on available days in the sprint, that is not a good way because you do not include the unexpected things" [OPO]

"[Sometimes we have a] tester estimating design tasks and a designer estimating test tasks. It is important to know whose estimation should be looked at". [Line mgr]

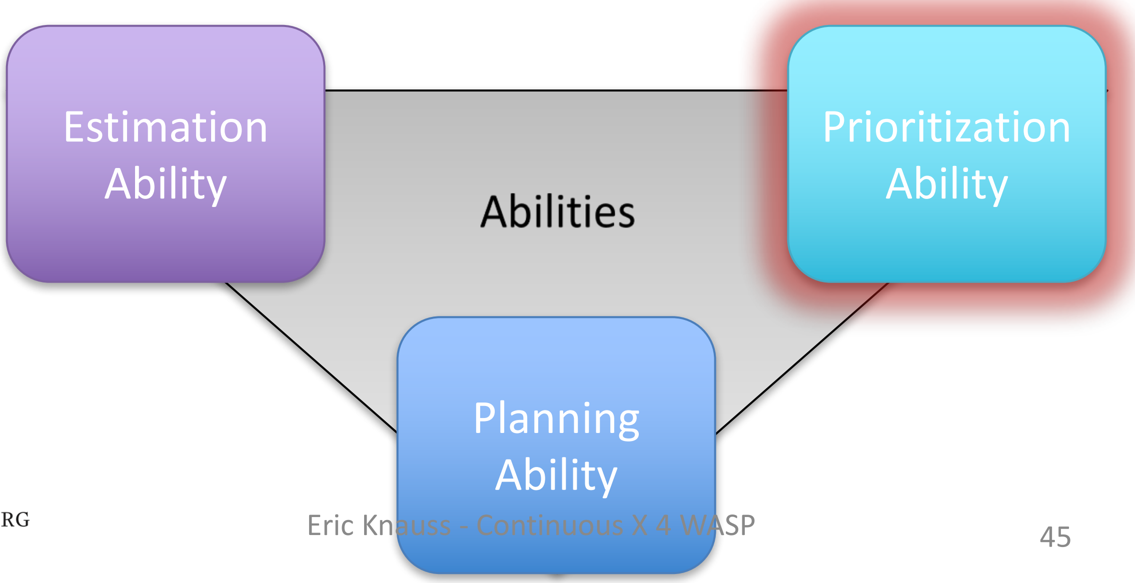
- Skeptical about estimations
- Need to monitor discussions
- *Estimate tasks that do not fit role*



Findings

“The challenge is if you have a lot of small backlog you are not in control at all be- cause if you have one common backlog and you decide on a program level, that is how we work [...] if not everything is visible on the common backlog program and only visible in the XFTs backlog then you maybe having a mismatch.” [OPO]

- Complex structure of product owners and backlogs
- Inconsistencies between backlogs
- Lack of transparency

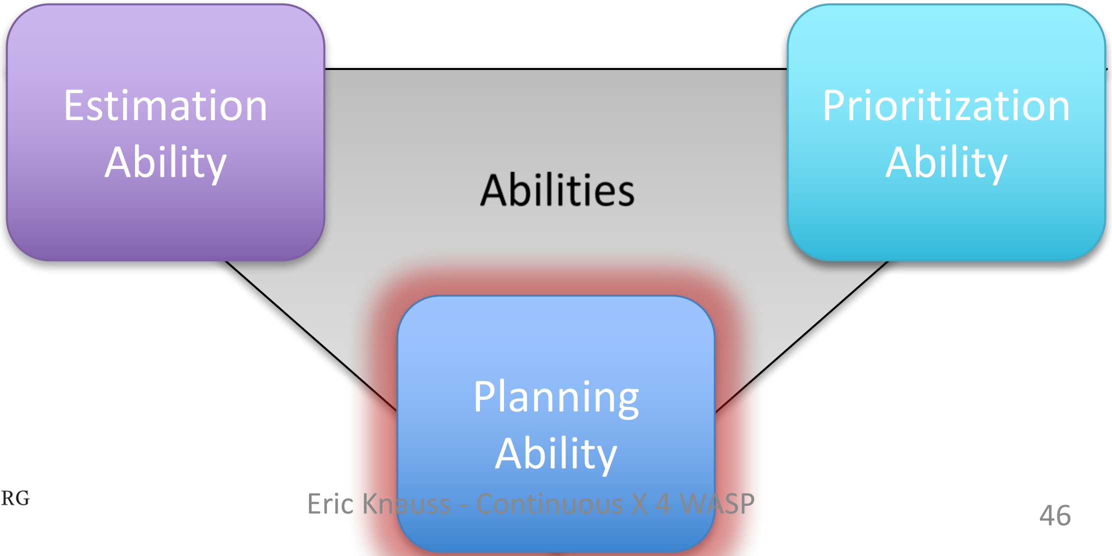


Findings

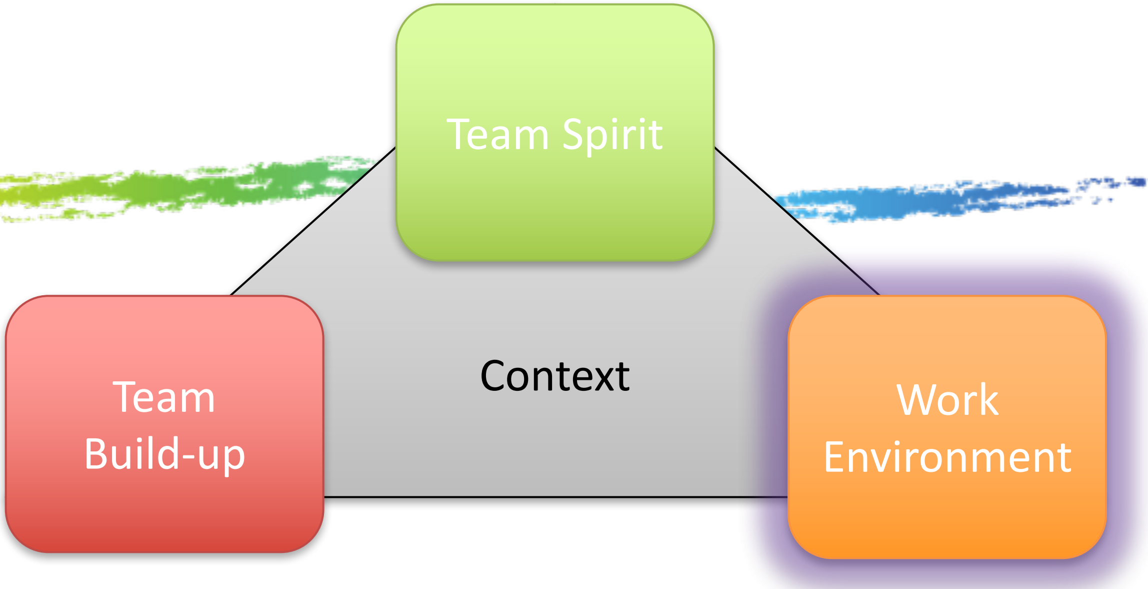
“... we have been working like this for two and half years now, and we are still struggling finding our roles”. *[OPO]*

”[planning is done by] me as the manager and perhaps with the help of the team leaders sometimes... [unclear] just how much should the team be involved in the planning phase and stuff like that..."
[Line mgr]

- Unclear requirements
- Unclear role of operational product owner
- Involvement of teams



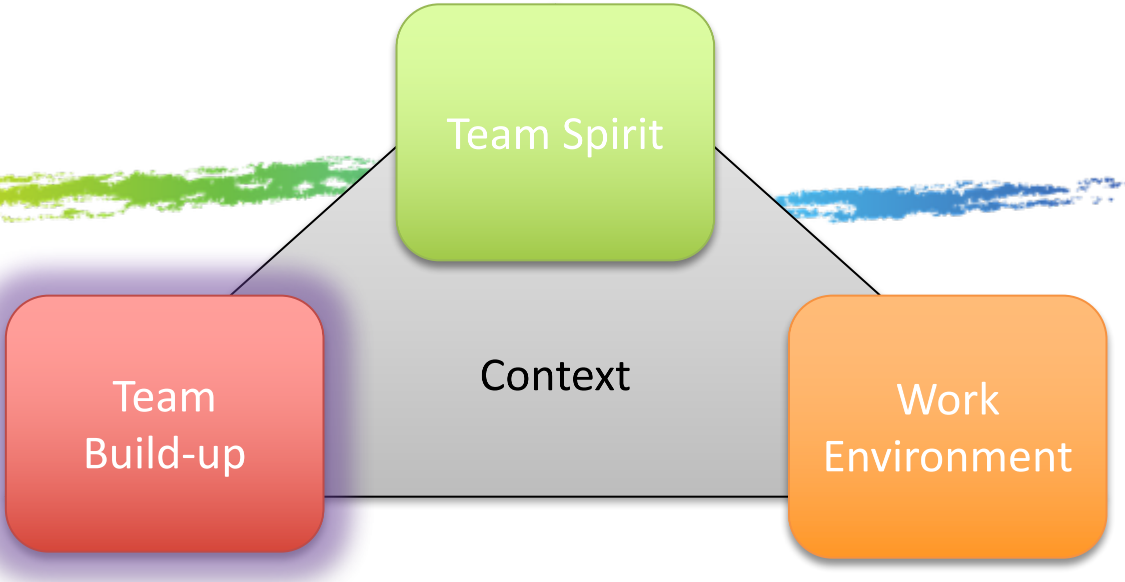
Findings



- Open Space
- Disturbing other teams
- By-passing operative product owner

"...we have scrum meetings in open office space [...]. You kind of get disturbed when other teams are having their scrum meeting in the open setting. It is better [if] every team has their different rooms." [Dev.]

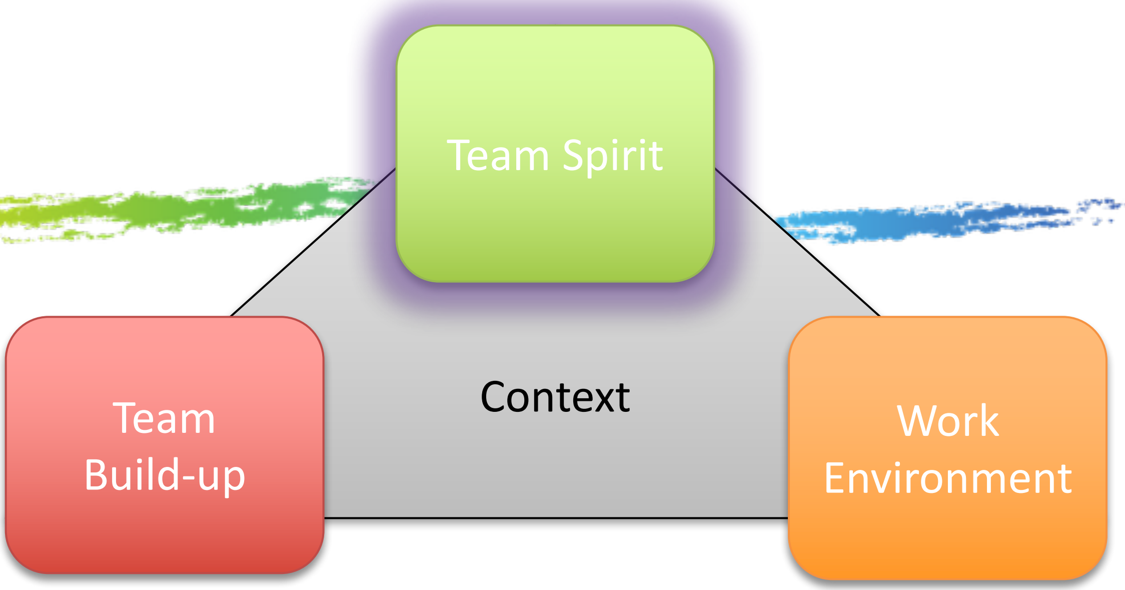
Findings



- Capabilities of team
- Moving team members
- Need for broader competence

“[Sometimes] operative product owner has one view and the team has another view [… It] can be a challenge to have the operative product owners to understand what capability a certain team has and he wants much more than they are capable of doing.”

Findings



- Shared concept of agile
- Team maturity

“moving somebody from one team to the other, then you are impacting the team spirit in both teams, you might go back to the team development stage when you get a new team member or lose a new team member”.

Findings

- Lack of suitable information channels
- Coordination meetings boring or too short
- Inadequate anatomy of features

Ceremony Agreement

"The biggest challenge I pick is the coordination of the feature portfolio, [...] on top of getting out features in our program fast and efficient, we need to collaborate on a portfolio basis to align the features over two programs".

[OPO]

Conclusion on Large-Scale Agile Planning

- Qualitative Model of what influences agile planning
  - Overview of key aspects of collaborative planning in large-scale agile development.
  - Large-scale agile planning not only depends on team abilities or skill, but also on the context in which those teams operate.
  - Ceremonies and practices on inter-team and inter-product level are currently missing and invite further research.

- Outlook: We encourage constructive research to provide improvement for one or several aspects. Our vision is a collection of best, or at least good, practices for each area in out model.

Kasauli, R.; Knauss, E.; Nilsson, A., Klug, S.: Adding Value Every Sprint: A Case Study on Large-Scale Continuous Requirements Engineering. In: Proc. Of 3rd Workshop on Continuous RE, Essen, Germany, 2017

Pre-Study: Adding value every sprint

- Benefits
  - *Internal:* Increase focus, quality of tests, feedback
  - *External:* Reduced distance to customer and risk, increased flexibility, faster learning
- Challenges
  - High effort to maintain quality
  - Risk of technical debt
  - Manage long-term perspective
- How to check value
  - Reviews, Sprint dem, Definition of Done, Te
- Suggested improvements
  - Be pro-active
  - Component guardians
  - Focus on process quality
  - Include team earlier
  - Improved checks of value
  - Transparency

Consequence to Continuous Delivery

- Impediments
  - Lack of shared understanding of customer value
- Bottlenecks / Disruptors
  - Distance to Customer
  - Lack of focus on Sprint goal
  - Lack of test infrastructure
- Catalysts / Enablers
  - Adding value every sprint
  - Definition of Done
  - User stories linked to requirements and tests

Agenda today

- Continuous X:
  - Integration, Delivery, Deployment, and DevOps – Terminology and Origin
  - In the context of Large-Scale System Development
  - Continuous Deployment Practices

- Testing Strategies for Continuous X for Embedded Systems

- The Role of Value, Features, and Requirements

Key takeaways:

- Fundamental
  - Agile values
  - Testing
- Advanced
  - User value
  - System knowledge
  - Architecture

Master thesis
The group maturity and technical debt
Case Study: Agile teams

- Group Development Questionnaire (Wheelan).
- SonarQube.
- Unstructured interview.

Nassiba El haskouri
guselhana@student.gu.se