Housekeeping
Terminology
Challenges
Principles
Practices
Team level
Program level
Portfolio level
State of the art
Wrapping up

– L9 –
Requirements Engineering for Agile Methods (or: Agile Requirements Engineering?)
DAT232/DIT285 Advanced Requirements Engineering

Eric Knauss
eric.knauss@cse.gu.se

October 14, 2025

Housekeeping
Terminology
Challenges
Principles
Practices
Team level
Program level
Portfolio level
State of the art
Wrapping up

Outline

1 Housekeeping
2 Terminology
3 Challenges
4 Principles
5 Practices
Team level
Program level
Portfolio level
6 State of the art
7 Wrapping up

8. Please name one thing that you liked and one thing that you wished for in this lecture.

Hide Answers | Show Names | 5/30 Students Answered

.

The difference between validation and verification was important to learn. I wished for different review-techniques, which I got.

perfect time to do the lec before starting the review. well directed.

liked - examples and diagram slides disliked - very vague and ambiguous questions/answers on socrative

good

Housekeeping

- Example exams are online
- We will again offer TA positions in the ARE course next year.
- If you are interested, make yourself known ***after the course grade is out***.

Learning Objectives

Knowledge	Skills	Judgement
K1 Identify a common RE challenge in a given software development context.	S1 Plan suitable RE practices in a team with respect to a given software development context.	J1 Assess new requirements engineering knowledge (challenge, principle, practice) and relate them to the framework in this course.
K2 Choose an appropriate RE practice in a given software development context.	S2 Effectively apply a suitable RE practice in a team in a given software development context.	J2 Suggest suitable actions to overcome a lack of requirements knowledge in a software development context.
K3 Compare suitability as well as advantages and disadvantages of given RE practices in a given software development context.	S3 Analyze the effect and quality of the outcome of a set of or individual RE practices in a given software development context.	J3 Consider inter-team, program level and social/ethical implications of a set of RE practices in a given software development context.
K4 Explain the current state of practice and research in requirements engineering.		J4 Critically assess the effectivity of a set of RE practices from the perspective of the student's master program (e.g. Software Engineering & Technology/Management, Interaction Design, Game Design, Data Science, ...)

Learning Objectives

Interpretation for ...Interpretation:

K1,K2,K3	Understand how agile ways of working affect RE challenges and practices
K4	Be aware of current research landscape of RE for Agile Development
S1-S3	Be able to plan, apply, and analyse the effect of RE practices in an agile context
J1-J3	Map your project experience and knowledge of the course so far against agile context.
J4	Reflect on how your background changes your view on RE in a typical development scenario. (which practices make sense? which do not?)

Housekeeping
Terminology
Challenges
Principles
Practices
Team level
Program level
Portfolio level
State of the art
Wrapping up

Outline

1 Housekeeping

2 Terminology

3 Challenges

4 Principles

5 Practices
- Team level
- Program level
- Portfolio level

6 State of the art

7 Wrapping up

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

Scrum

SCRUM Master

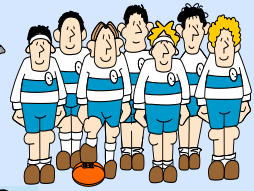
Others

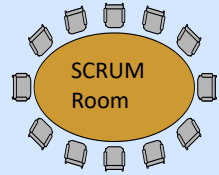
Product Owner

Product Backlog (prioritized)

SPRINT Backlog

SCRUM team 7+/-2





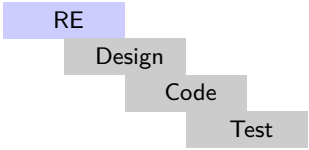
Increment

SPRINT: 30 days

Daily SCRUM
15 min.

Requirements Engineering, then and now

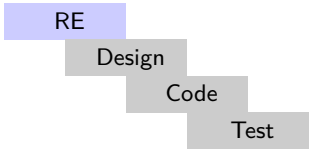
Then



Requirements engineering is a "waterfall phase" with specialists

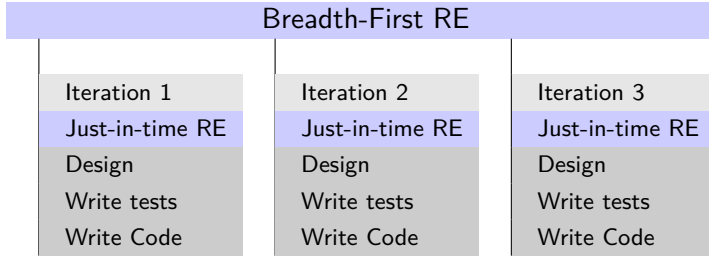
Requirements Engineering, then and now

Then



Requirements engineering is a "waterfall phase" with specialists

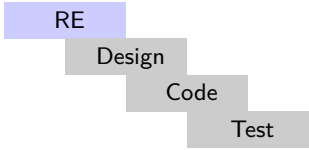
Now



Requirements are everybody's responsibility

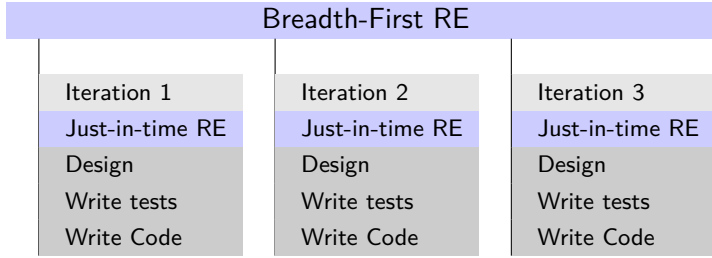
Requirements Engineering, then and now

Then



Requirements engineering is a "waterfall phase" with specialists

Now



Requirements are everybody's responsibility

Requirements Engineering:

a knowledge management problem

Agile RE in Research

Terminology

Just-in-Time RE and Breadth-First RE [Ernst and Murphy, 2012]

Requirements were [...] first sketched out with simple natural language statements [we refer to this as Breadth-First RE], and then fully elaborated during development, what we call just-in-time RE.

More elaboration on Just-in-time and Breadth-First RE found in [Niu et al., 2014].

Agile RE in Research

Terminology

Agile RE [Ramesh et al., 2010]

The term "agile RE" is used to define the "agile way" of planning, executing and reasoning about requirements engineering activities.

Agile RE practices from literature

- Face-to-face communication
- User stories
- Iterative requirements
- Extreme Requirements prioritization
- Change management
- Cross-functional teams
- Prototyping
- Test Driven Development
- Review meetings and acceptance tests
- Managing requirements change through constant planning

Face-to-face communication

- Facilitates customer involvement
- Challenges in establishing trust between customer and developer

Extreme Requirements prioritization

- Done at each step of cycle
- Unstable architecture

Managing requirements change

- Minimal documentation
- Inadequate architecture

Agile RE in Research

Terminology

Agile RE practices from literature

- **Face-to-face communication**
- User stories
- Iterative requirements
- **Extreme Requirements prioritization**
- Change management
- Cross-functional teams
- Prototyping
- Test Driven Development
- Review meetings and acceptance tests
- **Managing requirements change through constant planning**

Housekeeping
Terminology
Challenges
Principles
Practices
Team level
Program level
Portfolio level
State of the art
Wrapping up

Outline

1 Housekeeping
2 Terminology
3 Challenges
4 Principles
5 Practices
Team level
Program level
Portfolio level
6 State of the art
7 Wrapping up

Sentiments in literature

positive

+	Agile practices support RE	[Ramesh et al., 2010, Heikkilä et al., 2017, Bjarnason et al., 2011]
+	Agile practices resonate very well with developers	[Salo and Abrahamsson, 2008]
+	Success in scaling them up	[Salo and Abrahamsson, 2008]
+	Possible to implement in large organizations	[Lindvall et al., 2004]
+	Good support for knowledge sharing	[Lagerberg et al., 2013]
+	Traditional RE and agile go well together. Only difference: opinion about amount of documentation needed	[Paetsch et al., 2003]

Sentiments in literature

neutral

+/−	Agile methods *can address some classic communication challenges* in RE, but hard to ensure sufficient competence in XFT	[Bjarnason et al., 2011]
+/−	Requirements flow in large-scale agile: Increased flexibility, planning efficiency, communication efficiency. Problems with over-commitment, organizing system-level work, and growing technical debt.	[Heikkilä et al., 2017]
+/−	Agile RE: 8 new challenges, 17 challenges of traditional RE solved	[Inayat et al., 2015], which is the [AGRE] paper, discussed in a minute.

Sentiments in literature

negative

–	Difficult to manage reqts. in large-scale agile.	[Savolainen et al., 2010]
–	Hard to implement efficient RE	[Laanti et al., 2011, Wiklund et al., 2013, Chow and Cao, 2008]
–	Lack of empirical works on solutions and RE in relation to agile	[Inayat et al., 2015, Heikkila et al., 2015, Heikkilä et al., 2017]
–	Challenging to coordinate and to integrate into system development	[Eklund et al., 2014]
–	Definition of "Agile RE" is weak. Challenges: Customer representatives, prioritiz., techn. debt	[Heikkila et al., 2015]
–	Agile RE risks: neglect quality reqts, customer inability	[Ramesh et al., 2010]
–	RE and Agile do not go well together: Scenario focus and rejection of upfront X biggest damage that agile has caused to world	[Meyer, 2014], has been course book for agile course in our Master programmes.

Challenges with Agile and RE

Consider reading [AGRE], an excellent systematic literature review on the topic [Inayat et al., 2015]

- Challenges with traditional RE that Agile practices address
- New challenges with agile RE

Source: [AGRE / ISM+2015]

Table 7
Summary of challenges of traditional RE resolved by agile RE practices.

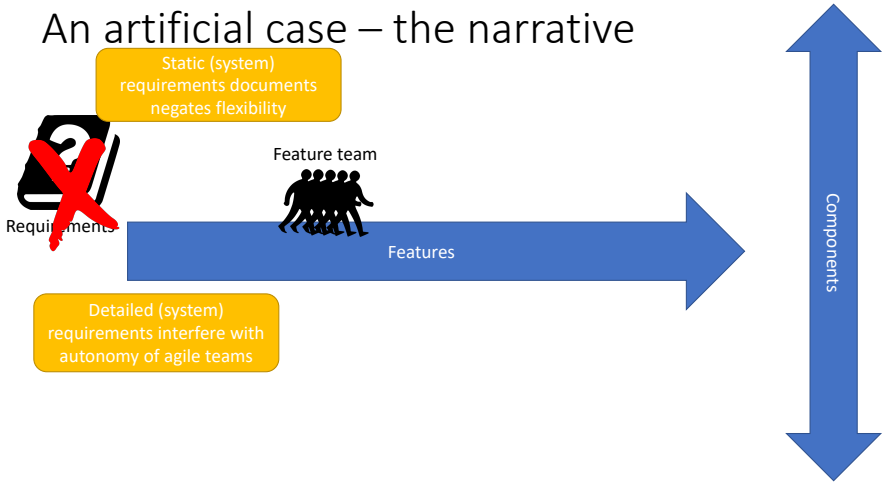
No.	Challenge	Practice	Description
1.	Communication issues (Bjarnason et al., 2011a; Carlson & Matuzic, 2010)	Frequent face-to-face meetings (Bang, 2007; Sillitti et al., 2005)	Agile RE promotes regular interaction with customer and among teams. It is the predominant method for eradicating communication gaps
		Collocated teams (Highsmith & Fowler, 2001)	Agile principles prefers collocated teams for better communication and collaboration
		Onsite customer (Cao & Ramesh, 2008; Lundh & Sandberg, 2002; Pichler et al., 2006)	Customer and development teams should be located in the same place to enhance informal communication, to enable timely feedback, to facilitate agreement, to develop ownership, and to create a sense of responsibility
		Alternate customer representations (Bjarnason et al., 2011a; Fraser, Mellon, Dunsmore, & Lundh, 2001; Hoda, Noble, & Marshall, 2011)	In industry, having business representatives to be present at the development site is expensive and impossible at times. RE alternatives such as proxy customers can serve the same purpose
		Cross-functional agile teams (Bjarnason et al., 2011a)	Cross-functional agile teams aid in the clarification and understanding of requirements
		Integrated RE process (Bjarnason et al., 2011a)	Locating RE process closer to development activities enhances developer's understanding and reduces communication lapses
2.	Overscoping (Bjarnason et al., 2011a)	One continuous scope flow (Bjarnason et al., 2011a)	In agile methods, developers receive a list of features that are constantly prioritised by the customer. Thus, the chance of having to repeat allocation in projects is reduced
		Gradual detailing (Bjarnason et al., 2011a)	Gradual detailing of requirements helps to reduce overscoping and contributes to a feasible scope
		Cross-functional teams (Bjarnason et al., 2011a)	The team can focus more on important features when sharing responsibilities and working closely together in a cross-functional structure
3.	Requirements validation (Carlson & Matuzic, 2010)	Requirements prioritisation (Racheva et al., 2010)	Customer continues with prioritisation requirements in every iteration; thus, less important requirements remain on hold
		Prototyping (Cao & Ramesh, 2008; Ramesh et al., 2010)	Prototyping helps in providing the customer with a blueprint of the product, and therefore helps in validating the requirements
4.	Requirements documentation (Bjarnason et al., 2011a)	User stories (Bjarnason et al., 2011a; Carlson & Matuzic, 2010)	User stories are precise and provide to-the-point explanation of user demands, prevent the need for maintaining long SRS documents as well as constant updating and traceability
		Face-to-face communication (Cao & Ramesh, 2008; Ramesh et al., 2010)	More face-to-face communication reduces ambiguities and the need for maintaining long documents
5.	Rare customer involvement (Carlson & Matuzic, 2010)	Requirements prioritisation by the customer (Racheva et al., 2010)	Prioritisation of the requirements for all iterations ensures, to a large extent that the customer goals will be met

Source: [AGRE]

Table 8
Summary of challenges of agile RE.

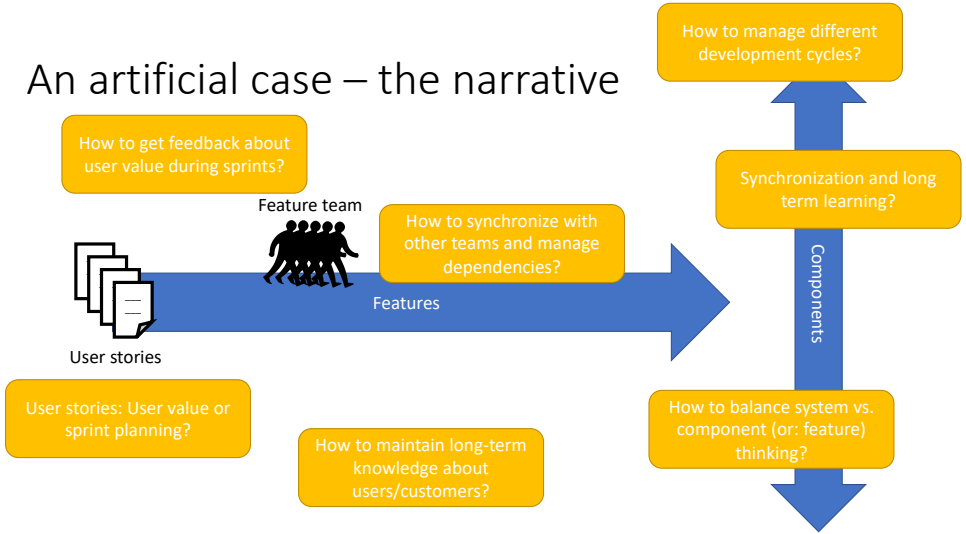
Challenge	Description	Impact	Solutions
Minimal documentation (Cao & Ramesh, 2008)	User stories and product backlogs are the only documents in agile methods (Zhu, 2009)	Traceability issues (Zhu, 2009)	
Customer availability (Ramesh et al., 2010)	Availability of customer for requirements negotiation, clarification and feedback	Increase in rework	Surrogate customers (Ramesh et al., 2010)
Inappropriate architecture (Ramesh et al., 2010)	Inadequate infrastructure can cause problems during later project stages	Increase in cost	Code refactoring (Berry, 2002)
Budget and time estimation (Cao & Ramesh, 2008)	Initial estimates of time and cost are changed substantially by a change in requirements in subsequent stages	Project delays	Frequent communication
		Over-budgeting	Accurate modelling of user story
Neglecting non-functional requirements (NRFs)	User stories only satisfy system/product features	System security, usability, performance at stake	NRF modelling approach (Farid & Mitropoulos, 2012b). The NORMATIC tool (Farid & Mitropoulos, 2012a)
Customer inability and agreement (Daneva et al., 2013; Ramesh et al., 2010)	Incomplete domain knowledge and in consensus among customer groups	Increase in rework	Creation of delivery stories to accompany user stories (Daneva et al., 2013)
		Increase in cost	Frequent communication Iterative RE (Ramesh et al., 2010)
Contractual limitations (Cao & Ramesh, 2008)	Fixed-price contracts do not allow changes	Increase in cost	
Requirements change and its evaluation	To find the consequences of requirements change	Increase in work delay	RE-KOMBINE framework (Ernst et al., 2013)

An artificial case – the narrative



[Knauss, 2019]

An artificial case – the narrative



[Knauss, 2019]

Conclusion

Challenges

→ `https://socrative.com`, REQENG, Question 1 and 2

Housekeeping
Terminology
Challenges
Principles
Practices
Team level
Program level
Portfolio level
State of the art
Wrapping up

Outline

1 Housekeeping
2 Terminology
3 Challenges
4 Principles
5 Practices
Team level
Program level
Portfolio level
6 State of the art
7 Wrapping up

State of the Art 3

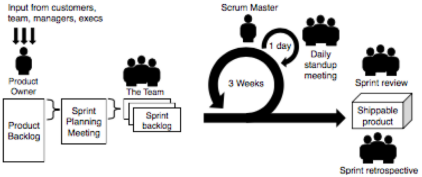


Fig. 1. Scrum Framework (adapted from Deemer and Benefield [28]).

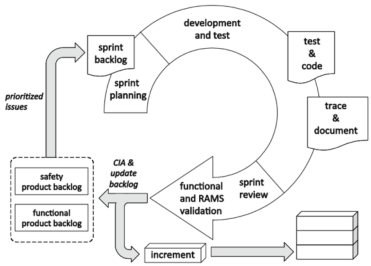


Fig. 2. The SafeScrum model

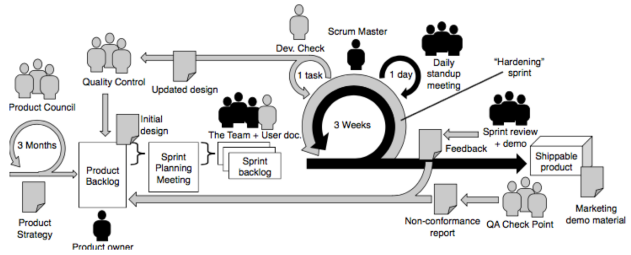


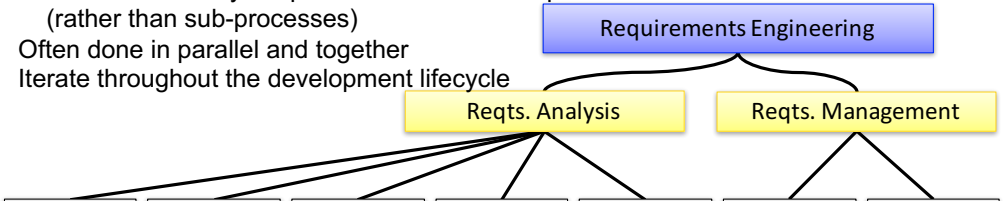
Fig. 4. R-Scrum: Regulated Scrum implementation at QUMAS.

Fig. 3. The tool-chain supporting the SafeScrum process

RE from Agile Perspective

Useful, if applied in a lightweight way

A number of loosely coupled activities and competencies
(rather than sub-processes)
Often done in parallel and together
Iterate throughout the development lifecycle



Src: DaimlerChrysler, Dagstuhl-Seminar 1998

UNIVERSITY OF GOTHENBURG

CHALMERS UNIVERSITY OF TECHNOLOGY

Then

Now

RE
Design
Code
Test

Breadth-First RE

Iteration 1	Iteration 2	Iteration 3
Just-in-time RE	Just-in-time RE	Just-in-time RE
Design	Design	Design
Write tests	Write tests	Write tests
Write Code	Write Code	Write Code

Requirements engineering is a "waterfall phase" with specialists

Requirements are everybody's responsibility

Requirements Engineering:

a knowledge management problem

"when?"

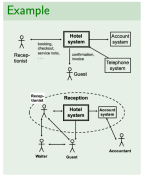
What is RE?

Scopes

What is Requirements Engineering?

- Business case (or: goal-level) requirements
- User (or: domain-level) requirements
- System (or: product-level) requirements
- Design-level requirements

Example



"perspective?"

"what?"

Activities

What is Requirements Engineering?

A number of loosely coupled activities and competencies (rather than sub-processes)
Often done in parallel and together
Iterate throughout the development lifecycle

Requirements Engineering

Reqts. Analysis | Reqts. Management

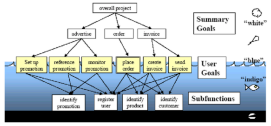
Elicitation	Interpretation	Negotiation	Documentation	V & V	Change Mgmt	Tracing
• Identify Stakeholders and other Reqt src • Capture needs, ideas, etc. • Goal analysis	• Identify requirements • Structure / refine • Operationalize and make concrete	• Identify dependencies • Identify + solve inconsistencies • Prioritize	• Document reqts, current state, assumpt. • Create baseline • Administrative attributes	• Correct requirements? • Requirements correct?	• Manage change requests • Manage versions of reqts • Propagate changes	• Within reqts • Pre-Tracing (reqt to src) • Post-Tracing (reqt to change set, test)

Src: DaimlerChrysler, Dagstuhl-Seminar 1998

"abstraction?"

Abstraction levels

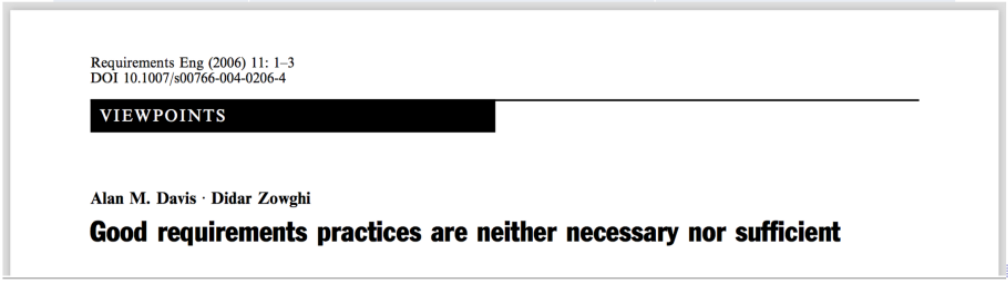
What is Requirements Engineering?



[Cockburn, 2001]

[Gorschek and Wohlin, 2006]

Do we need requirements engineering?



Requirements Eng (2006) 11: 1–3
DOI 10.1007/s00766-004-0206-4

VIEWPOINTS

Alan M. Davis · Didar Zowghi

Good requirements practices are neither necessary nor sufficient

Do we need requirements engineering?

Requirements Eng (2006) 11: 1–3
DOI 10.1007/s00766-004-0206-4

VIEWPOINTS

Alan M. Davis · Didar Zowghi

Good requirements practices are neither necessary nor sufficient

Context	Agile	RE

Do we need requirements engineering?

Requirements Eng (2006) 11: 1–3
DOI 10.1007/s00766-004-0206-4

VIEWPOINTS

Alan M. Davis · Didar Zowghi

Good requirements practices are neither necessary nor sufficient

Context	Agile	RE
Startup	(-) MVP and Learning over agile practices!	(+) Addresses main problem: No clear problem to solve
Small mobile/web app		

Do we need requirements engineering?

Requirements Eng (2006) 11: 1–3
DOI 10.1007/s00766-004-0206-4

VIEWPOINTS

Alan M. Davis · Didar Zowghi

Good requirements practices are neither necessary nor sufficient

Context	Agile	RE
Startup	(-) MVP and Learning over agile practices!	(+) Addresses main problem: No clear problem to solve
Small mobile/web app		
Large-scale software dev.		
Large-scale system dev.		

Question

→ `https://socrative.com`, REQENG, Question 3

Do we need requirements engineering?

Requirements Eng (2006) 11: 1–3
DOI 10.1007/s00766-004-0206-4

VIEWPOINTS

Alan M. Davis · Didar Zowghi

Good requirements practices are neither necessary nor sufficient

Context	Agile	RE
Startup	(-) MVP and Learning over agile practices!	(+) Addresses main problem: No clear problem to solve
Small mobile/web app	(+) Ideal case	(-) Agile practices sufficient to cover RE
Large-scale software dev.	(+) But hard to implement, see SAFe / LESS frameworks	(+) Need to align work of several teams
Large-scale system dev.	(?) Very hard to implement, but tempting. HW may not lend itself well.	(+) Need to align work of different domains, suppliers, etc.

Housekeeping
Terminology
Challenges
Principles
Practices
Team level
Program level
Portfolio level
State of the art
Wrapping up

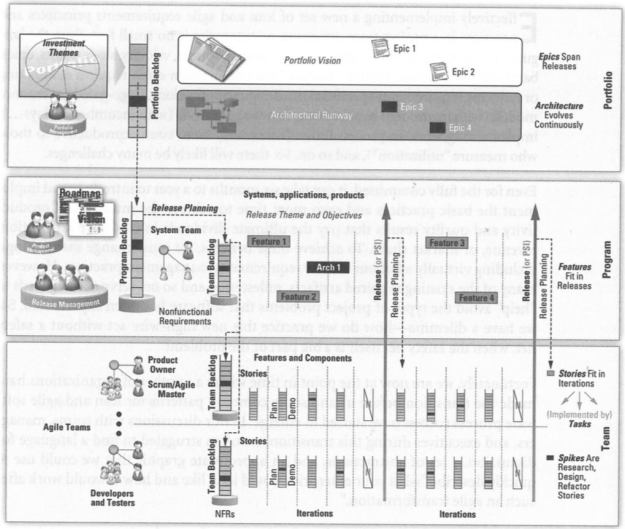
Outline

1 Housekeeping
2 Terminology
3 Challenges
4 Principles
5 Practices
Team level
Program level
Portfolio level
6 State of the art
7 Wrapping up

Practices

- But how to do it in practice?
- Let's take a look into the textbook [Leffingwell, 2011]
- This is, by the way, the RE flavour used in SAFe (scaled agile framework)

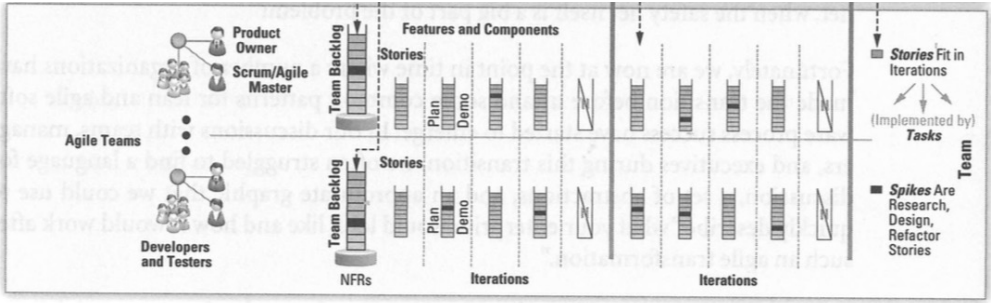
The "Big Picture" of Agile Requirements



[Leffingwell, 2011]

Team Level [Leffingwell, 2011]

- Roles:
  - Product Owner
  - Agile Coach
  - Developers / Testers
- Artifacts
  - User stories and Tasks



User stories

- Card
  - "As a <role> I can <activity> so that <business value>"
  - 1-3 sentences in total
  - Memorable token, summarizes intent, represents a more detailed requirement, whose details remain to be determined
- Conversation
  - A discussion between team, customer (representative), product owner, and other stakeholders
  - Necessary to determine the more detailed behavior required to implement the intend
  - A user story includes the promise of this conversation!
- Confirmation
  - Acceptance test: How the customer will determine that the story has been implemented to satisfaction
  - Conditions of satisfaction

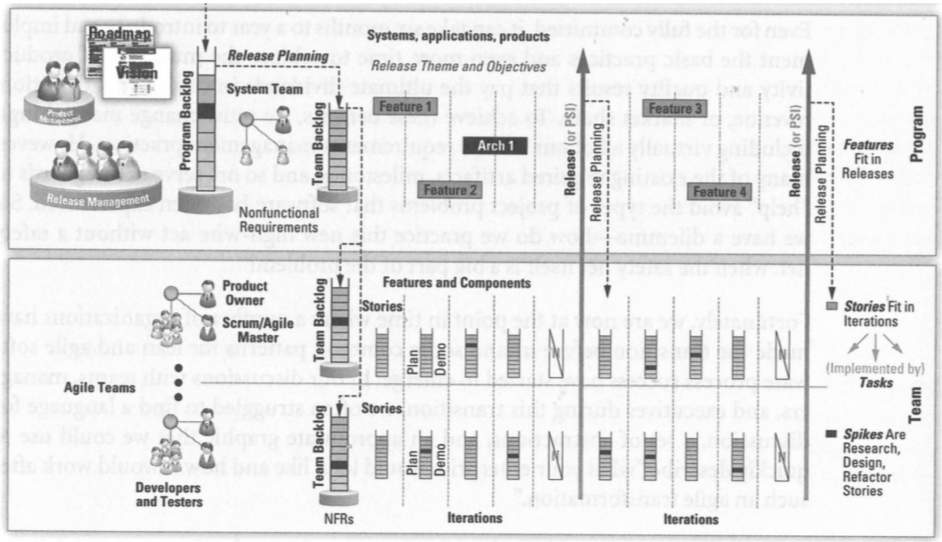
INVEST in User Stories

- Independent
- Negotiable... and negotiated
- Valuable
- Estimable
- Small
- Testable

Team Level Requirements Activities

- Spikes
  - Do research, learn about technical feasibility, explore alternatives
  - Source of requirement updates!
- Stakeholders
  - Levels of involvement: informed, consulted, partners, in control
- Agile estimating and velocity
  - Story points: an abstraction from time as measure of effort
  - Often Fibonnacci numbers: 1, 2, 3, 5, 8, 13
  - Planning poker
  - Source of requirements updates!
- Testing
  - Tests complement user stories
  - Source of requirements updates!

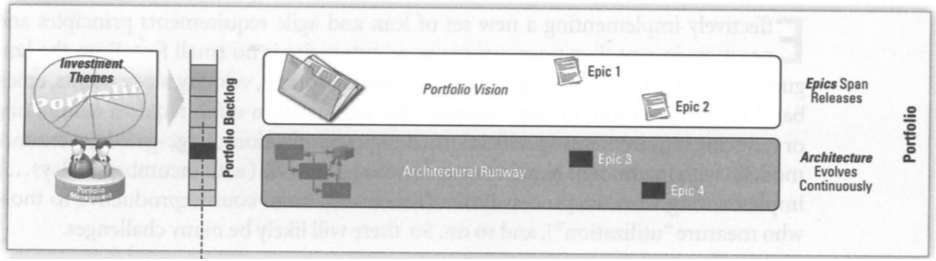
Program Level



Program

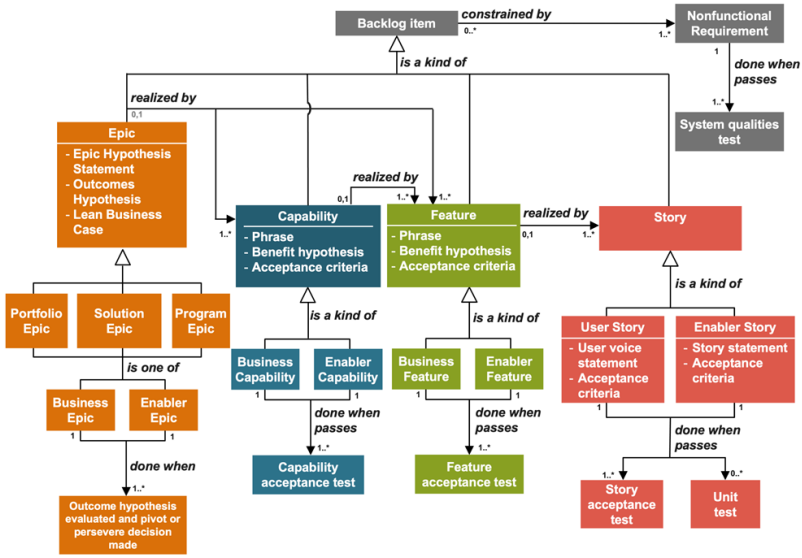
- Manage Non-functional requirements
  → They are cross-cutting
- Most of requirements analysis takes place here
- Roadmapping and Distributed Prioritization
  → Difficult, need to interfere with team autonomy
- Agile Release Train
  → Coordinate a set of teams towards a set of releases
  - Frequent periodic planning
  - Teams apply common iteration lengths
  - Intermediate, global, objective milestones established
  - Release increments are available at regular intervals
  - System-level hardening to reduce technical debt, allow for release-level validation and testing
  - Team infrastructure must trail ahead (licenses, platform APIs, SDKs, ...)

Portfolio



- Epics span multiple releases
- Architecture is defined here
- Both inform roadmapping
- Both rely on high level requirements engineering

SAFe - Requirements Information Model



Source: https://www.scaledagileframework.com/safe-requirements-model/

Conclusion

Practices

→ `https://socrative.com`, REQENG, Question 4-6

Housekeeping
Terminology
Challenges
Principles
Practices
Team level
Program level
Portfolio level
State of the art
Wrapping up

Outline

1 Housekeeping
2 Terminology
3 Challenges
4 Principles
5 Practices
Team level
Program level
Portfolio level
6 State of the art
7 Wrapping up

State of the art

Well, at least things we are currently working on...

- Boundary objects as a way to identify a minimum set of system documentation [Wohlrab et al., 2019]
- Agile methods for safety-critical systems [Kasauli et al., 2018]
- Continuous integration and deployment viewpoints in system architecture [Pelliccione et al., 2017]
- A managers' perspective on how RE contributes to development speed [Ågren et al., 2019]
- Using git, markdown, review or pull-request based workflows to manage system requirements [Knauss et al., 2018]

TReqs Pitch

- *Objective:*
  - TReqs offers lightweight tooling to manage requirements in agile system development.
- *Philosophy:*
  - TReqs empowers agile teams to manage requirements together with changes of code and test.
  - This ensures transparency, consistency, scalability, and speed.
- *Offer:* The core of TReqs is available as open source. Based on this, we offer:
  - **Integrating** TReqs into a specific company's requirements strategy.
  - **Developing** a company's requirements strategy if needed.
  - **Adjusting** TReqs to match specific needs
  - **Training** for agile teams, product owners, system managers.

TReqs: Context and Motivation

- Engineering complex systems increasingly done agile and continuous
- Agile (SW) teams will discover new or wish to update existing requirements
- Any system level role that owns requirements will become a bottleneck
- Agile teams avoid working with outdated requirements (don't read/don't update)
- System development looses a critical coordination mechanism

TReqs core idea

- Distinguish high-level customer/market/problem-focused requirements from system/solution-focused requirements
- Give agile (SW) teams ownership of system requirements in a scalable way
  - Bring the requirements into the tools that agile teams work with
  - Integrate reviews of requirements changes into the quality assurance workflows of agile teams
  - Derive reports for system level roles
  - Support DevOps vision of infrastructure-as-code
- Result: scalability, speed, and real control through up-to-date requirements
  - Instead of: illusion of control by gatekeeping requirements that are not used

Getting started: TReqs demonstrator

- Assume that Alice and Bob want to create a function that translate latin numbers to roman numbers
- The start with a high-level requirement
  - (which they store in a markdown file for now, since those are nicely layed out and well integrated in their git based development environment (gitlab or github))

```
1 # Req−1: Convert arabic to roman
2
3 The function takes an integer i and returns a String that represents i as a roman number.
```

Basic workflow

- Alice wants to get started. Working in an agile way, she aims to use TestFirst

```
1  assertEquals("I", convert(1));
2  ...
3  convert(int i) {return "I";}
4  ...
5  # Req-1: Convert arabic to roman
6
7  The function `convert` takes an integer i and returns a String that represents i as a roman number.
8
9  ## Req-1.1: Convert 1 to I
10
11 For i = 1, `convert` shall return `I`.
```

- Alice pushes these changes to her development branch and creates a merge request
- Bob reviews the change, sees that test, function, and requirement are updated and consistent
- The change is merged into the main branch
- Alice and bob continue with the number 2 to 10 in parallel

Create requirements

- Now, to support this scenario, we want to add some support.
  - Example, adding a link between the unit test and the Requirement 1
- If we want to offer tool support, we need to
  - define a unique ID (otherwise, Alice and Bob creating each a Requirement 1.1 at the same time will cause chaos)
  - make it easy for the tool to define where a requirement or test starts and ends
- With treqs create, we can conveniently create syntactically correct treqs elements, that are still rendered as markdown elements

```
<treqs-element id="b1cd3dba866a11ebbdfcc4b301c00591" type="requirement">
## Req-1.1: Convert 1 to I

For i = 1, `convert` shall return `I`.

<treqs-link type="parent" target="72f032c0866d11ebac03c4b301c00591" />
</treqs-element>
```

List requirements

- Now we have requirements in markdown files in a very useful form
  - They are easy to read for humans, either in the markdown file or in the rendered preview
  - They are easy to read for computers
- We can list requirements based on several criteria, which helps to keep an overview in large projects

```
knauss$ treqs list
| UID | Type | Label | File | Line |
| :---- | :---- | :---- | :---- | :---- |
| 72f032c0866d11ebac03c4b301c00591 | requirement | # Req-1: Convert arabic to roman | requirements/system-requirem
| b1cd3dba866a11ebbdfcc4b301c00591 | requirement | ## Req-1.1: Convert 1 to I | requirements/system-requirements.m
| 54ed41e2867111eb91e5c4b301c00591 | unit-test |     * ## arabic2roman test | src/se/treqs/example/numconv/NumConv
```

The output of treqs list can be rendered as a markdown table:

UID	Type	Label	File	Line
72f032c0866d11ebac03c4b301c00591	requirement	# Req-1:	requirements/system-	

Trace requirements

- Through the UID, we can also create and explore tracelinks from tests to requirements
- In javadoc or other comments just in front of automated tests, we can add tracelinks such as

```
1  /**
2   * <treqs-element id="54ed41e2867111eb91e5c4b301c00591" type="unit-test">
3   * ## arabic2roman test
4   *
5   * This test checks requirement Req-1
6   * <treqs-link type="required-by" target="72f032c0866d11ebac03c4b301c00591" />
7   * </treqs-element>
8   */
9  @Test
10 public void arabic2roman() {
11    assertEquals("I", conv.convert(1));
12    assertEquals("II", conv.convert(2));
13    assertEquals("III", conv.convert(3));
14 }
```

Check requirements

- Several checks are possible. Most important: are critical tracelinks set

```
knauss$ treqs check
| Error location | Error | File | Line |
| :—— | :—— | :—— | :—— |
| Element b1cd3dba866a11ebbdfcc4b301c00591 | Unrecognised link type parent within element of type requirement. | re
| Element 54ed41e2867111eb91e5c4b301c00591 | Element has an unrecognized type: unit—test | src/se/treqs/example/nu
treqs check exited with failed checks.
```

Error location	Error	File	Line
Element b1cd3dba866a11ebbdfcc4b301c00591	Unrecognised link type parent within element of type requirement.	requirements/system-requirements.md	9
Element 54ed41e2867111eb91e5c4b301c00591	Element has an unrecognized type: unit-test	src/se/treqs/example/numconv/NumConverterTest.java	2

Requirements as part of a commit

- By working in an agile, test-driven way, we now have commits to bundle changes of software, tests, and requirements.

```
knauss$ git status
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   requirements/system-requirements.md
    modified:   src/se/treqs/example/numconv/NumConverter.java
    modified:   src/se/treqs/example/numconv/NumConverterTest.java
```

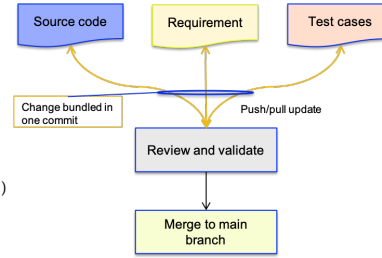
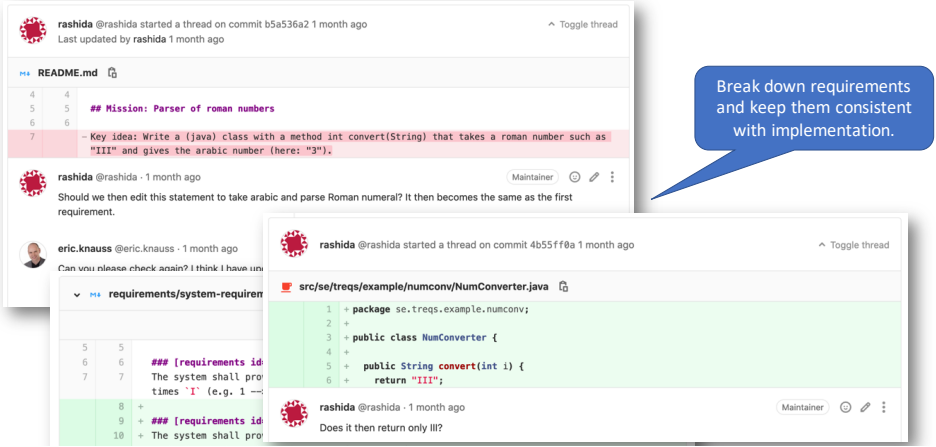


Figure 1: TReqs workflow

Review requirements



Modeling support

- Write UML models as text
- Use github to version control and merge
- Use plantuml to convert to png
- Use treqs to create png references
- TReqs can be extended to allow clickable links in models and tracing of model elements

Req-1: Convert arabic to roman

The function `convert` takes an integer i and returns a String that represents i as a roman number.

```
@startuml convert-example-use-case
:User: – (convert arabic to roman)
@enduml
```

Req-1.1: Convert 1 to I

Competition

- Agile illusion claims that user stories, automated tests, and coding standards are sufficient
  - Does not scale for complex systems / multiple teams
  - Does not allow to trace customer requirements to system properties (automated tests perform poorly in that role)
  - Regulatory needs are almost impossible to cover without sacrificing agility
- Classic requirements tools (Word, Excel, ...)
  - Do not grow with the needs of a startup/company
  - Very limited traceability and versioning
- Traditional requirements tools (Doors, RequisitPro, ...)
  - Licensing costs
  - Do not integrate into SW development tools
  - Do not intend massive parallel evolution of requirements
  - Do not integrate well with git and continuous deployment
- Git based tools (e.g. doorstop)
  - Do not provide the same level of conceptual depth
  - Start from wrong granularity for tracing

Benefits

- Text based
  - Easy to start or transition to
  - No vendor lock-in
- Integrates in git
  - Very powerful support for versioning and continuous integration/deployment
  - Supports standard technology stack of agile teams
- Infrastructure-as-code
  - extensible, customizable
- Open source ($\rightarrow$ https://gitlab.com/treqs-on-git/treqs-ng)

Housekeeping
Terminology
Challenges
Principles
Practices
Team level
Program level
Portfolio level
State of the art
Wrapping up

Outline

1 Housekeeping
2 Terminology
3 Challenges
4 Principles
5 Practices
Team level
Program level
Portfolio level
6 State of the art
7 Wrapping up

Todo

- Attend L10 - Ethics and RE (Thu, Oct-16, 13:15-15:00). Panel style. Bring questions.
- Create, submit, rehearse Project Presentation.
- Deadlines
  - Project Presentation (Mon, Oct-20, 8:00am)
  - Release R3 (Mon, Oct-27, 8:00am)
  - Digital-hall exam (Thu, Oct-30, 8:30)

→ `https://socrative.com`, REQENG, Question 7-8

References I


Ågren, S. M., Knauss, E., Heldal, R., Pelliccione, P., Malmqvist, G., and Bodén, J. (2019).
The impact of requirements on systems development speed: A multiple-case study in automotive.
Requirements Engineering (REEN), 24(3):315–340.


Bjarnason, E., Wnuk, K., and Regnell, B. (2011).
A case study on benefits and side-effects of agile practices in large-scale requirements engineering.
In *Proc. of 1st WS on Agile Reqts. Eng.*


Chow, T. and Cao, D.-B. (2008).
A survey study of critical success factors in agile software projects.
Journal of Systems and Software, 81:961–971.


Eklund, U., Olsson, H. H., and Strøm, N. J. (2014).
Industrial challenges of scaling agile in mass-produced embedded systems.
In *Proc. of Int. WS on Agile Methods. Large-Scale Dev., Refactoring, Testing, and Estimation*, pages 30–42.


Ernst, N. and Murphy, G. (2012).
Case studies in just-in-time requirements analysis.
In *Proc. of Int. Workshop on Empirical Requirements Eng. (EmpiRE)*, pages 25–32, Chicago, IL, USA.


Heikkila, V. T., Damian, D., Lassenius, C., and Paasivaara, M. (2015).
A mapping study on requirements engineering in agile software development.
In *Proc. of 41st Euromicro Conf. on Softw. Eng. and Advanced Applications (SEAA ?15)*, pages 199–207.


Heikkilä, V. T., Paasivaara, M., Lasssenius, C., Damian, D., , and Engblom, C. (2017).
Managing the requirements flow from strategy to release in large-scale agile development: a case study at ericsson.
Empirical Software Engineering, pages 1–45.

References II

Inayat, I., Salim, S. S., Marczak, S., Daneva, M., and Shamshirband, S. (2015).
A systematic literature review on agile requirements engineering practices and challenges.
Computers in human behavior, 51:915–929.
See [AGRE] on Canvas.

Kasauli, R., Knauss, E., Kanagwa, B., Nilsson, A., and Calikli, G. (2018).
Safety-critical systems and agile development: A mapping study.
In *Proc. of Euromicro SEAA*.

Knauss, E. (2019).
The missing requirements perspective in large-scale agile system development.
IEEE Software, 36:9–13.

Knauss, E., Liebel, G., Horkoff, J., Wohlrab, R., Kasauli, R., Lange, F., and Gildert, P. (2018).
T-reqs: Tool support for managing requirements in large-scale agile system development.
In *Proc. of Int. Requirements Engineering Conference (RE)*.
Tool Demo Track.

Laanti, M., Salo, O., and Abrahamsson, P. (2011).
Agile methods rapidly replacing traditional methods at nokia: A survey of opinions on agile transformation.
Information and Softw. Techn., 53::276–290.

Lagerberg, L., Skude, T., Emanuelsson, P., Sandahl, K., and Ståhl, D. (2013).
The impact of agile principles and practices on large-scale software development projects: A multiple-case study of two projects at ericsson.
In *Proc. of Int. Symp. on Empirical Software Engineering and Measurement*, pages 348–356.

References III

Leffingwell, D. (2011).
Agile Software Requirements.
Addison-Wesley.

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., May, J., and Kahkonen, T. (2004).
Agile software development in large organizations.
Computer, 37:26–34.

Meyer, B. (2014).
Agile! The Good, the Hype and the Ugly.
Springer.

Niu, N., Bhowmik, T., Liu, H., and Niu, Z. (2014).
Traceability-enabled refactoring for managing just-in-time requirements.
In *Int. Requirements Eng. Conf. (RE)*, pages 133–142, Karlskrona, Sweden.

Paetsch, F., Eberlein, A., and Maurer, F. (2003).
Requirements engineering and agile software development.
WETICE, 3:308.

Pelliccione, P., Knauss, E., Heldal, R., Ågren, S. M., Piergiuseppe, M., Alminger, A., and Borgentun, D. (2017).
Automotive architecture framework: the experience of volvo cars.
Journal of systems architecture.

Ramesh, B., Cao, L., and Baskerville, R. (2010).
Agile requirements engineering practices and challenges: an empirical study.
Information Systems Journal, 20:449–480.

References IV

Salo, O. and Abrahamsson, P. (2008).
Agile methods in european embedded software development organisations: a survey on the actual use and usefulness of extreme programming and scrum.
IET software, 2:58–64.

Savolainen, J., Kuusela, J., and Vilavaara, A. (2010).
Transition to agile development: rediscovery of important requirements engineering practices.
In *18th Int. Req. Eng. Conf.*, pages 289–294.

Wiklund, K., Sundmark, D., Eldh, S., and Lundqvist, K. (2013).
Impediments in agile software development: An empirical investigation.
In *Proc. of Product-Focused SW Process Impr.*, pages 35–49.

Wohlrab, R., Pelliccione, P., Knauss, E., and Larsson, M. (2019).
Boundary objects and their use in agile systems engineering organizations.
Journal of Software: Evolution and Process, 31:1–24.