

SWEN - Software Engineering Network – Donnerstag 06. Mai. 2010

Agile Requirements Engineering

Blaise Rey-Mermet, © EVOCEAN GmbH, 2010



My background



Blaise Rey-Mermet
EVOCEAN GmbH

Agile Processes & Requirements Engineering
Dipl. Natw. ETHZ, Master of Technology Enterprise IMD

Email: blaise.rey-mermet@evocean.ch

Executive Roles

- Dept. Head - Requirements Management & Engineering Competence Center
- Director and Dept. Head in R&D groups and consulting groups

Coach

- Requirements Management Coach
- Process Improvement Coach
- Agile Coach

Software Engineer

- Requirements Engineering
- Software Developer & Architect
- Project Manager
- Scrum Master

Agenda

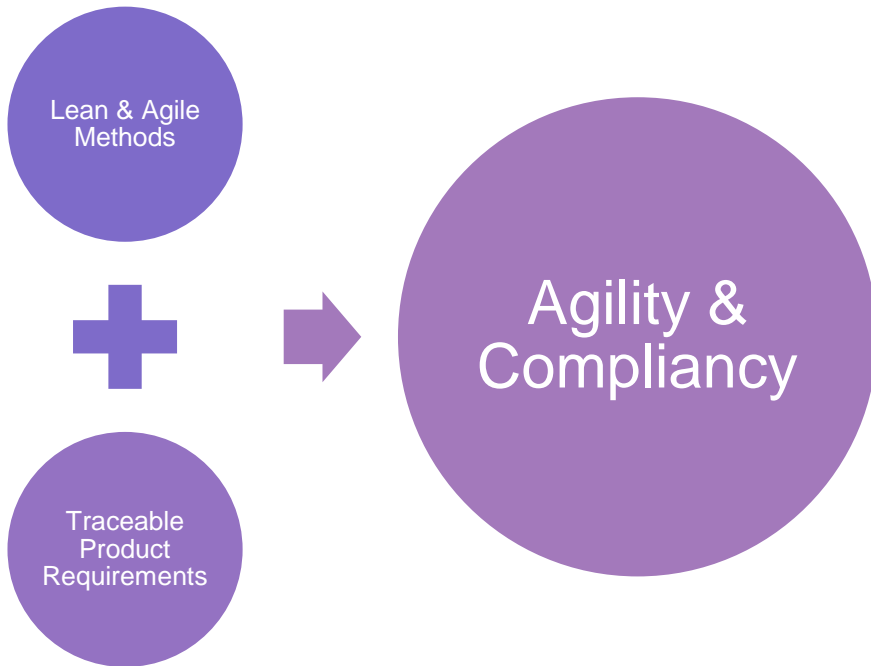
Principles of Agile Requirements in Scrum

Writing Agile Requirements

Balancing Agility and Compliance

Establishing Agile Requirements in the Organization

The acceleration of change makes your customers requirements unpredictable: keep them agile!



Agile methods

- **Experimentation** must be built into the development process
- Complicated problems need to be broken down to be understandable
- The development process must take into account that it is **not possible to fully define the product at its inception**

Requirements management

- **Sufficient information** is required on the product to be built
- Requirements are **traceable** and prioritized

Principles of Agile Requirements in Scrum

Manifesto for Agile Software Development

<http://agilemanifesto.org/>

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.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

The principles of Lean Manufacturing applied to product development

- eliminate waste
- build-in quality
- create knowledge
- defer commitment
- deliver fast
- respect people
- optimize the whole

Waste in requirements is anything that does not add value (from a customer point of view):

- functionality that does not work
- functionality that the user does not understand

Ref: Implementing Lean Software Development by Mary & Tom Poppendieck

Principles of agile requirements

- **Use a customer driven approach:** the customer specifies and prioritize the product development
- **Value driven:** let the customer states his purpose
- **Embrace change:** requirements are not contracts
- **Get immediate feedback:** by letting the customer use the new features
- **Write just in time requirements** and add details as needed, and not upfront
- **Write just enough requirements:** stop when the product shows that the desired features are understood
- **Enforce participatory design:** specify the customer needs - the development team has the freedom to innovate when creating a product that satisfies them

Just enough and just-in-time requirements

Write the detail of requirements when

- the requirement is about to be designed
- there is a business / customer value in specifying the detail
- there is a regulatory need to get the requirement document now

Look for alternatives:

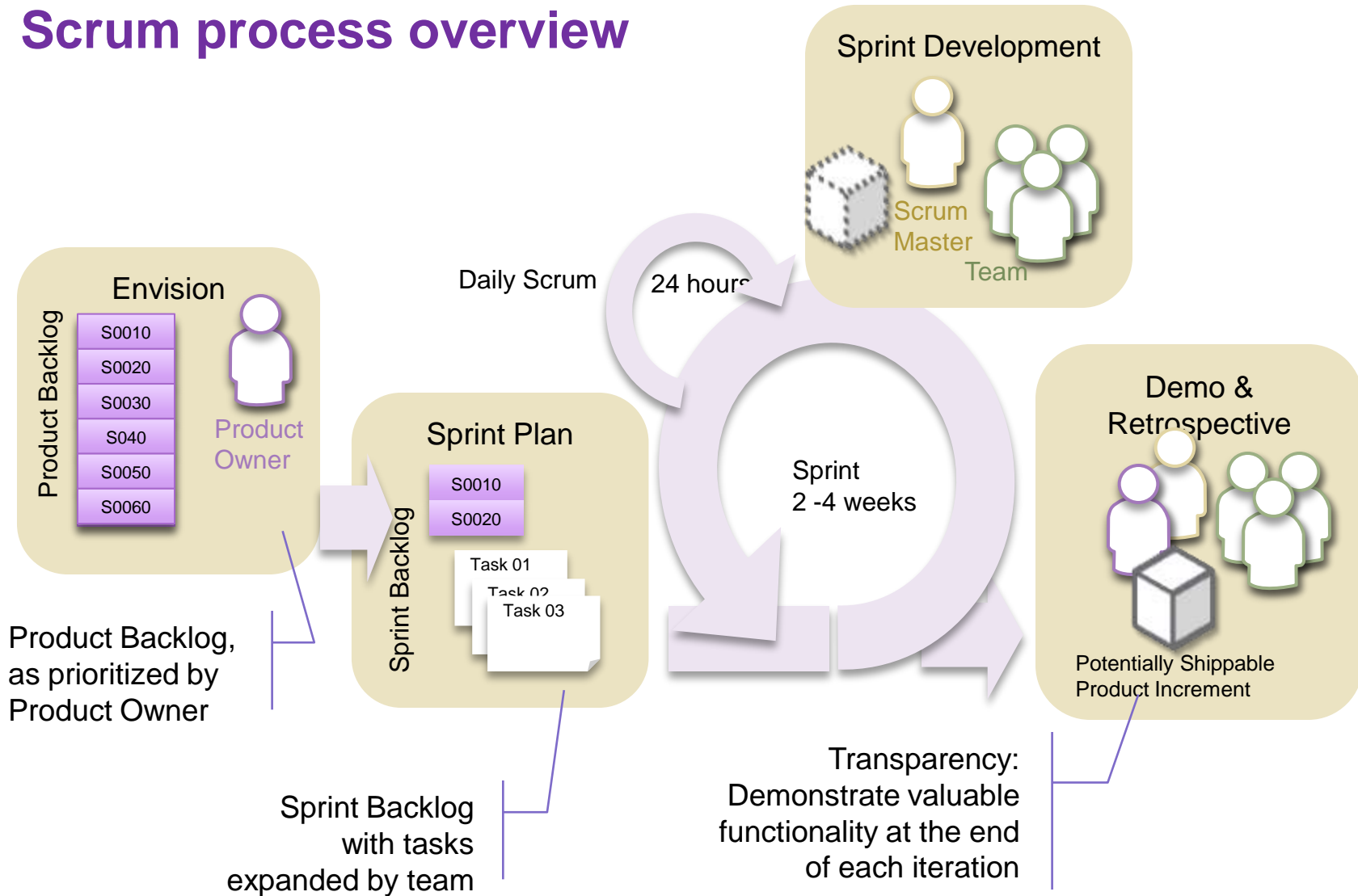
- write the details later
- look for alternatives to narrative text. For example sketches, unit tests...

How Scrum enforces agile requirements

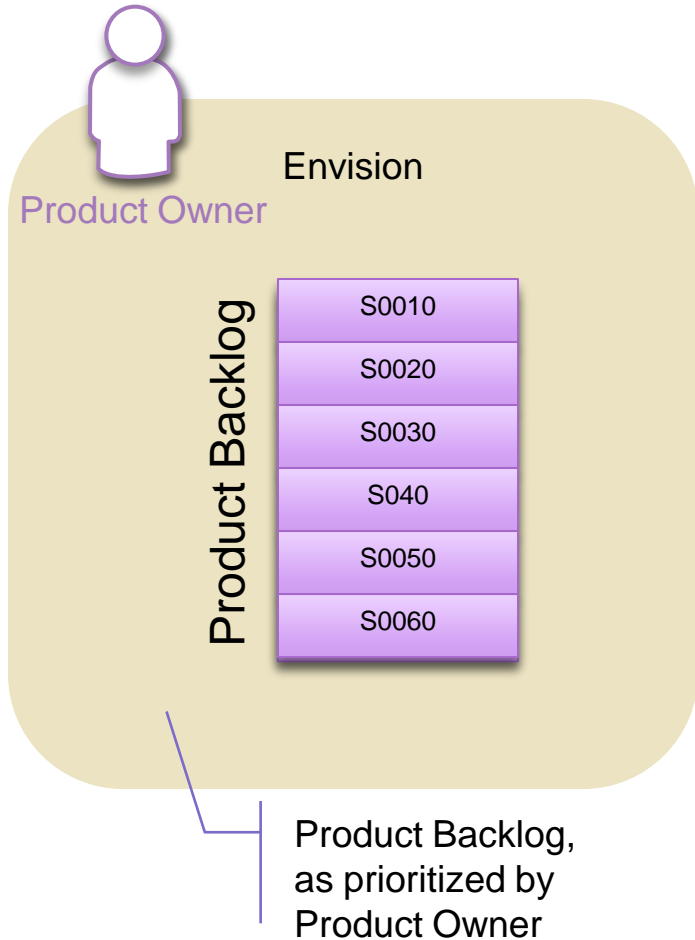
Scrum characteristics

- Scrum is an agile process that allows us to **focus on delivering** the highest business value in the shortest time.
- **Self-organizing teams:** The business sets the **priorities**. Teams **self-organize** to determine the best way to deliver the highest priority features.
- **Iterative:** It allows us to **rapidly and repeatedly inspect** actual working software. Every two weeks to a month anyone can see **real working software** and decide to release it as is or continue to enhance it for another sprint.
- **Reduced number of artifacts:** Requirements are captured in a single list of “product backlog”.
- **Reduced number of roles** (product owner, scrum master, team).
- No specific engineering practices prescribed.

Scrum process overview



Scrum roles: Product Owner



Responsibilities

- Enforce a shared vision
- Define the requirements
- Decide on release date and content
- Be responsible for the profitability of the product (ROI)
- Manage and prioritize the backlog according to market value
- Accept or reject work results

Writing Agile Requirements

Write User Stories because they encourage the formulation of the purpose

Story 01: *As a patient, I want clear indication of the implant operating mode, so I can easily read it when I am tired or confused.*

Priority 4 [10 Story Points]

A **User Story** is a brief description of something the system needs to do and that will be valuable either to a user or to an owner.

Stories are usually expressed in a standard form, so that each story has value to the users of the product, such as
As a <user role> **I want to** <feature> **so I can** <value>

This is an **active** rather than **passive** form to write a requirement (rather than “the system should..”).

The User Story describes...

Who (user role)

What (feature)

Why (value)
The value clarifies
why a feature is
useful

Story 01: *As a patient, I want*
clear indication of the implant
operating mode, *so I can* easily
read it when I am tired or
confused.

Priority 4 [10 Story Points]

..but what if we do need to document the details during later sprints?

A story card with a conversation.

Story 01: *As a patient, I want clear indication of the implant operating mode, so I can easily read it when I am tired or confused.*

Priority 4 [10 Story Points]

(Thomas) up to 7 modes are possible depending on the implant type.

Write a conversation on the card to add details.

Why conversations? Stories are not contractual obligations. Conversations avoid the false impression of precision.

Why are User Stories agile requirements?

User Stories:

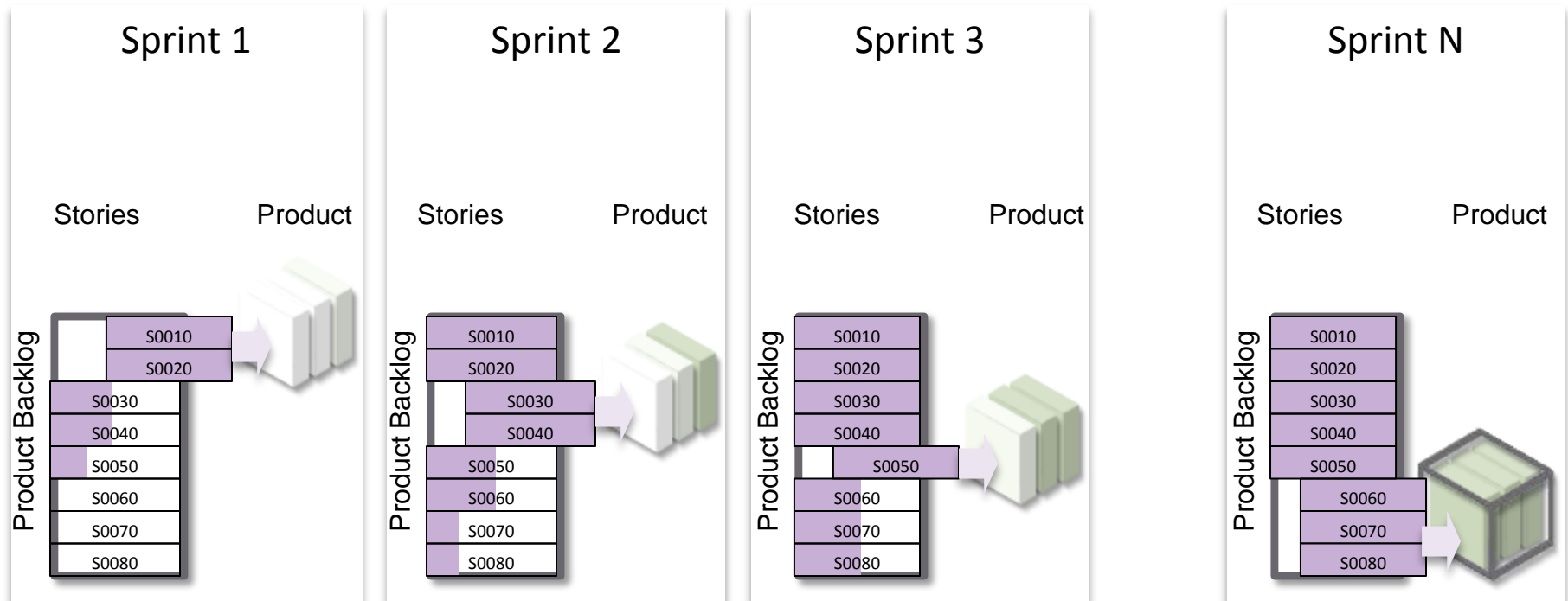
- encourage the formulation of the purpose
- emphasize verbal communication
- are comprehensible by all stakeholders
- have the right size for planning
- work for iterative development.
- encourage deferring details
- encourage participatory design

Drawbacks to User Stories:

- it is difficult to organize many small stories
- their traceability needs to be documented
- do not replaces documentation for efficient communication in the long term

Are detailed requirements allowed in agile projects?

- Write detail requirement only for the requirements that are under development.
- The complete requirements are established at the end of the last sprint.
- Changes in requirements that are not yet developed or under development have no impact.



Stories with high priorities are to be understood first.

Product Backlog (as prioritized by the Product Owner) sufficiently understood

↑
Understandability

S01: A user gets the actual data at startup (Auto-refresh)
Prio **1** [4 Pt]

S08: Create compilable application skeleton
Priority **2** [10 Pt]

S12: As a new user I want to display the temp. forecast f in a simplest possible way so I can easily access them.
Prio **3** [5 Pt]

started
& done
stories

in discussion

S04: Set up continuous integration system
Prio **4** [4 Pt]

S02: Display current temperature in a simplest possible way
Prio **5** [10 Pt]

S06: Set up the web server for serving weather data
Prio **6** [15 Pt]

S05: A user can change the location
Prio **7** [5 Pt]

prioritized

S07: Set up the web server for serving weather data
Prio **8** [8 Pt]

S11: A user can change the location
Prio **9** [5 Pt]

S01: Set up continuous integration system
Prio **10** [20 Pt]

S09: Create compilable application skeleton
Prio **11** [10 Pt]

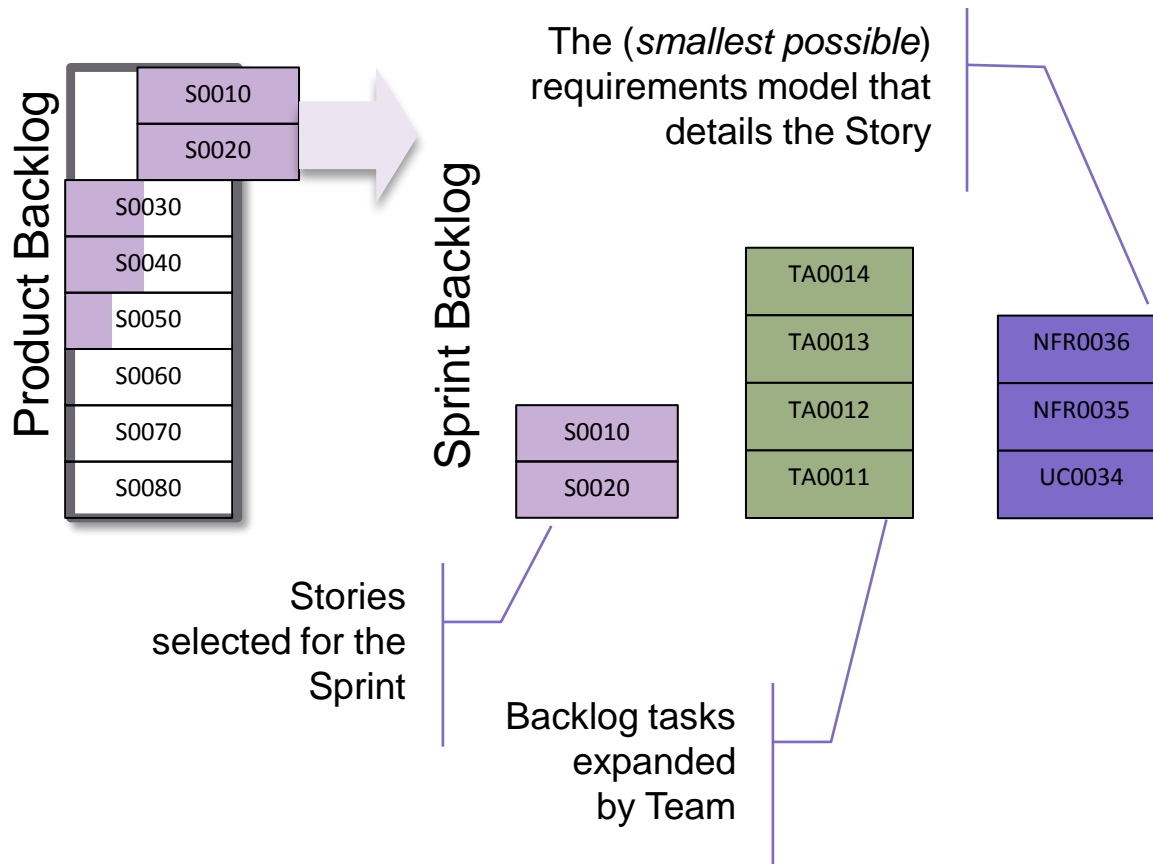
S10: Set up continuous integration system
Prio **12** [5 Pt]

accepted
stories

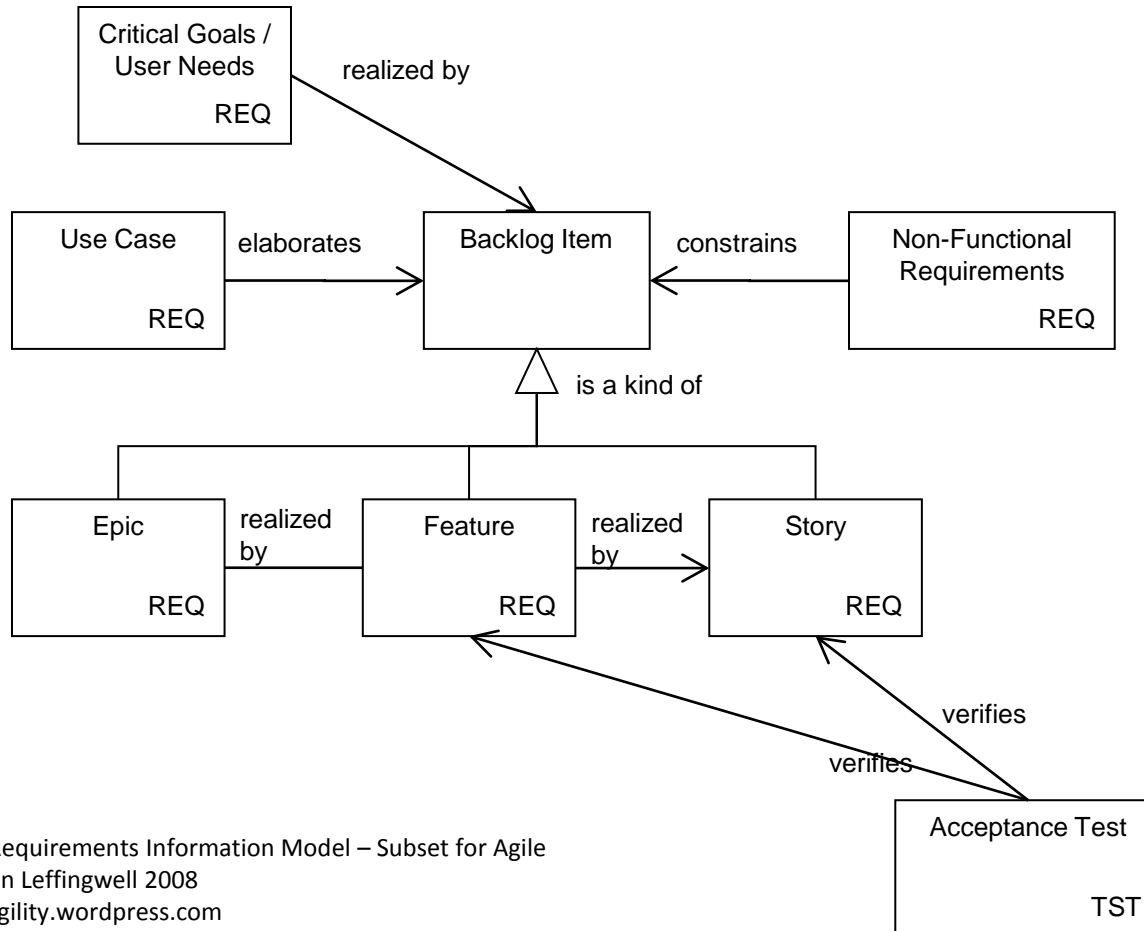
S13: A user gets a display of the sample temp. data according to the current location
Prio **13** [20 Pt]

S03: A user can display the actual snow coverage according to a given ski resort
Prio **14** [5Pt]

Detail the requirements with Use Cases and Non-Functional Requirements

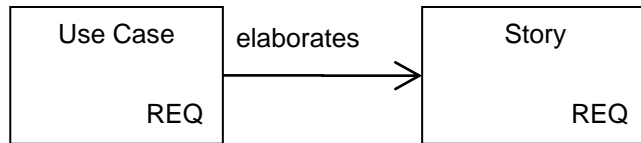


An agile requirements model (adapted from D.Leffingwell)



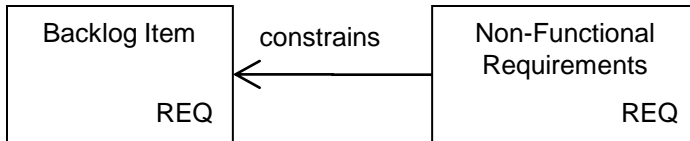
See : Agile Enterprise Requirements Information Model – Subset for Agile Project Teams - by Dean Leffingwell 2008
www.scalingsoftwareagility.wordpress.com

Write Use Cases when you need to describe a complete interaction



- Stories and Use Cases have different **scope**: Stories are kept small – Use Cases describe a complete interaction.
- Use Cases may **last longer**.
- There is **no need for an agreement** on Stories.
- Use Cases are more prone to include details of the User Interface.
- Use Cases can be used **to clarify a (set of) Stories that describes sequential functional behavior**

Add Non-Functional Requirements as constraints on Stories



Non-Functional Requirements

- specify the quality (quality in its broadest sense) an application or a system must provide to the users.
- are used to express the “how well” (i.e. in what quality);
 - performance, security, reliability
 - norms, standards
 - design constraints
- also need to be verified by Test Cases.

Add constraints on Stories

A Story Card with constraints

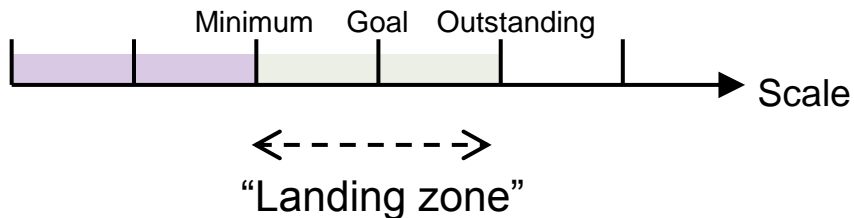
Story 01: *As a patient, I want* clear indication of the implant operating mode, *so I can* easily read it when I am tired or confused.

Priority 4 [10 Story Points]

NFR_1-5
(Environmental Constraint)
The control unit display shall be clearly legible under an ambient luminance.

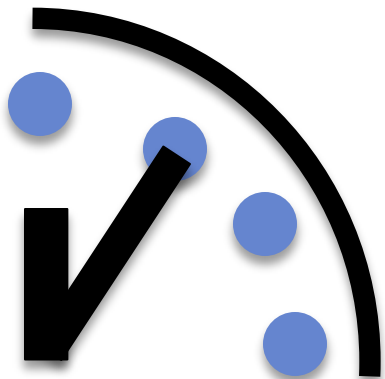
- You can put constraints on Story Cards on separated constraints cards for the system.
- Acceptance Tests can be written to ensure the constraint is not violated.

Use “Planguage” keywords to detail the goal and the level of achievement required



Planguage is an informal, but structured, keyword-driven language to formulate constraints

- **Scale:** The measurement units used to quantify the statement (e.g. “mean time between failure”).
- **Meter:** A practical method for measuring and testing on a defined scale.
- **Minimum:** level required to avoid failure.
- **Goal:** The level at which good success can be claimed.
- **Outstanding:** A future desired level, under defined [time, place, event] conditions, which is designed to challenge people to exceed Goal levels.



They are many more Planguage keywords – use the keywords that add value to your requirement – no more, no less.

Reference: Tom Gilb - Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage- August, 2005 - www.gilb.com

Document assumptions and risks for critical Stories

Story 01: *As a patient, I want* clear indication of the implant operating mode, *so I can* easily read it when I am tired or confused.

Priority 4 [10 Story Points]

(Thomas) Up to 7 modes are possible depending on the implant type.

- **Assumptions:** All assumptions or assertions that could cause problems if untrue now or later.
- **Risks:** Anything that could cause malfunction delay, or other negative impact.

NFR_1-5

(Environmental Constraint)

The control unit display shall be clearly legible under an ambient luminance.

Assumptions:

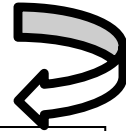
- 1) ambient luminance range: 100 to 1000 lux.
- 2) Viewing distance: 20cm to 60cm (for alarm signals, specific requirements shall apply).

Continue until there is “just enough” details to start the design.

A story card with acceptance tests.

Story 01: *As a patient, I want clear indication of the implant operating mode, so I can easily read it when I am tired or confused.*

Priority 4 [10 Story Points]



Try it in horizontal screen mode

Try it in ambient luminance range: 100 to 1000 lux.

Try it for viewing distance: 20cm to 60cm

Additional expectations of Users can be captured as **acceptance tests**.

You can write how to measure the required functionality, on which scale, and what the expected results will be, to make it testable and clarify the Users expectations.

... then review the Stories with the team and validate the product features

Review the Stories to ensure that there are **conforms to the customer's expectation of the product and the team's expectation**

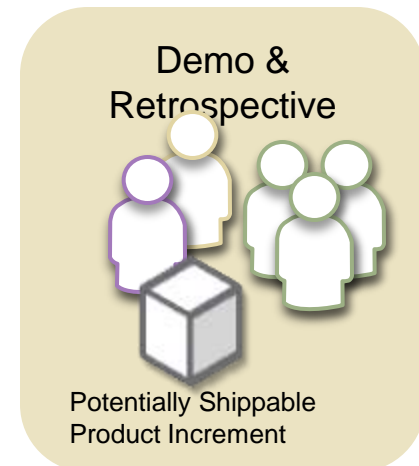
Verify the following **Stories characteristics:** (INVEST model)

- Independent
- Negotiable
- Valuable
- Estimable
- Small
- Testable

Validate the (potentially shippable) product to ensure that the product **fulfills the goals of the customers** (not the requirements)

The Sprint review

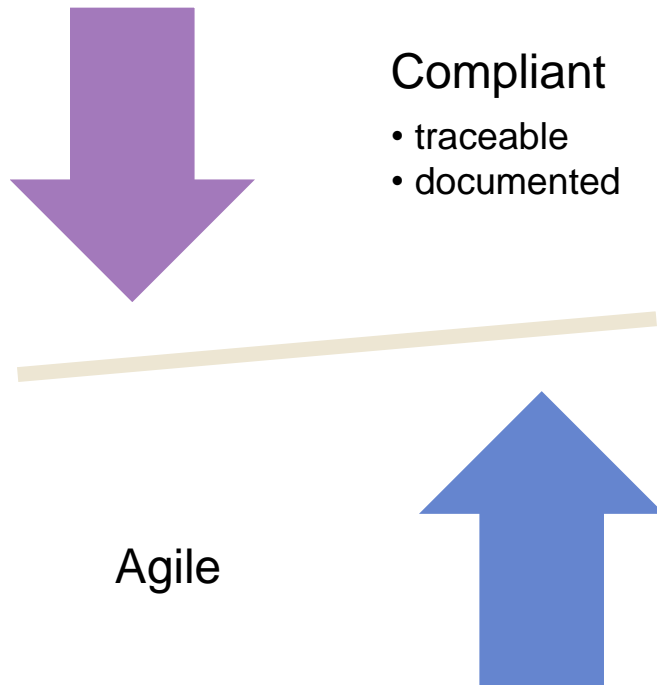
- The team presents what is accomplished during the Sprint
- Explain / demo the new features
- Let the Product Owner actually use the product.



Balancing Agility and Compliance

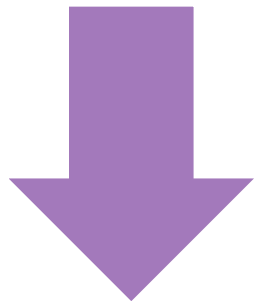
- Compliance Constraints for Requirements Management
- CMMI and agile Requirements
- IEC 62304 and agile Requirements
- Traceability of Stories

Compliance to regulations requires a defined process...



- That the development **process is documented and followed**
- **Best practices** to minimize product risks and maximize reliability
- That the product is **completely specified**
- That the product is **traceable** with the various levels of requirements

Compliance to regulations requires a defined process, ...agility makes it responsive to changes



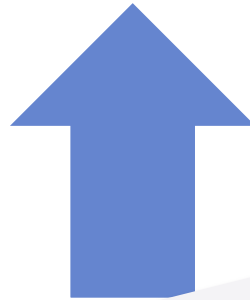
Compliant

- traceable
- documented



Agile

- responsive to change
- small steps
- creative



✓ Agile projects tend to be more disciplined -they follow less rules and teams are self-organized.

Agile processes better support aspects which former waterfall processes failed to address:

- **Experimentation** and **feedback** from lessons learned must be built-in.
- **Human creativity is not a linear process.** Many aspects of the development are covered simultaneously.
- **Complex problems** must be broken down in small steps to be managed.

Example: apply a CMMI® model to agile/lean development organizations

CMMI

- CMMI (Capability Maturity Model Integration) is a **process model** for system and software engineering
- CMMI defines 5 Maturity Levels: a roadman to establish good practices in organization (not just in teams)
- CMMI does not implies a waterfall process model. CMMI says “what” to achieve, not “how”
- CMMI applies to teams of all sizes
- It will reduce your workload, not blow it up

Agile

- Agile **defines a set of principles** and practices, not a process model
- The “right” documentation is required
- Customers never remember what they agreed to – even if they do understand what you’ve asked for. Some level of record helps.

Ref. <http://www.sei.cmu.edu/cmmi/> It is developed by the [Software Engineering Institute \(SEI\)](#) of the Carnegie Mellon University. - CMMI, CMM, and Capability Maturity Model are registered in the U.S. Patent and Trademark Office.

Map agile equivalent to the typical CMMI® evidences

CMMI: typical direct evidence in Requirements Definition (RD)

SG1 Elicit Customer Needs

- Business Needs Document
- Requirements Screening Criteria
- Customer Requirements
- Business Requirements

SG2: Develop Product Requirements

- Product Requirements
- Product-Component Requirements
- Interface Requirements
- Requirements Sign-Off

(SG = Specific Goal)

Agile Equivalent



- Product Backlog
- Stories
- Acceptance Tests on Stories

- Sprint Backlog
- Models
- Traces between Product and NFR
- Architecture Stories

Replace CMMI practices by lean/agile practices when agile is better suited to reach the goal.

Map agile equivalents to the CMMI® practices

CMMI Requirements Management (REQM)

Specific Goal: *Requirements of projects, product or product components are managed, and the consistencies are identified between those requirements, project plans and work products.*

- SP 1.1 Obtain an understanding of requirements
- SP 1.2 Obtain commitment to requirements
- SP 1.3 Manage requirements changes
- SP 1.4 Maintain bi-directional traceability of requirements
- SP 1.5 Identify inconsistencies between project work and requirements

(SP = Specific Practice)

Agile Equivalent

In an agile context, requirements are documented in some other form than in plan-driven development projects, e.g. with user story cards or a user story database.

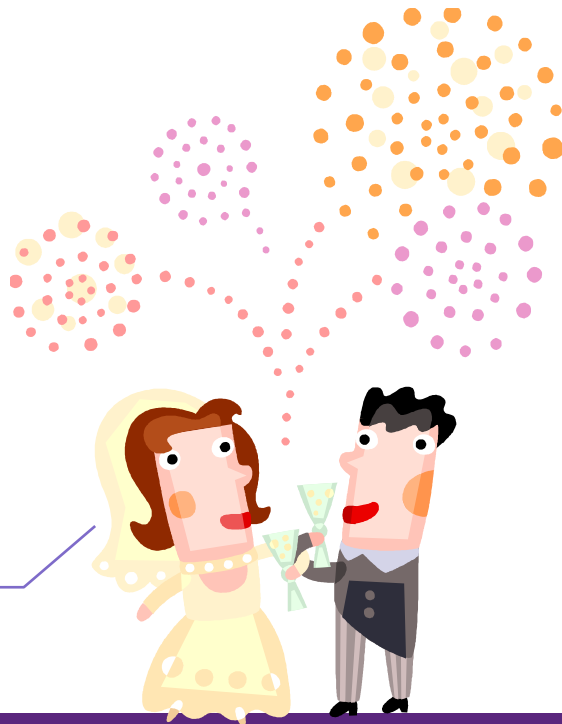
- SP 1.1 & 1.2 The Sprint planning meeting attests that the User Stories are discussed with all stakeholders.
- SP 1.3 Add Configuration Management for changes in Stories that are already used in the sprint.
- SP 1.4: The traceability of requirements can be supported by keeping a record of the earlier Stories, tasks and functional tests.
- SP 1.5: Review meetings attest this. Inconsistencies can be also addressed by the agile practice of writing codes specifically to meet test cases.

Enhance lean/agile practices when insufficient

CMMI® and agile are complementary

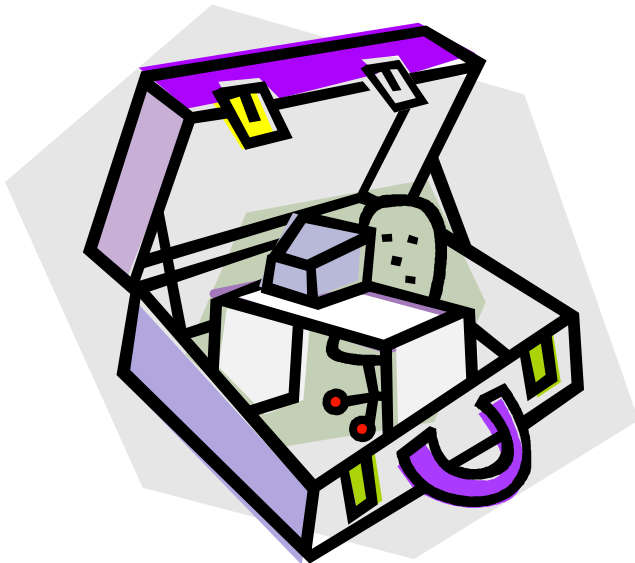
- Perceived differences are often in approach, not substance.
- You can implement a CMMI-compliant software development process that is agile and brings you the repeatability and predictability offered by CMMI.
- Use agile practices as alternatives to the CMMI “typical” artifacts (and train your Lead Appraiser)!
- ...the next release of CMMI (V1.3, expected in November 2010) will include agile material.
- To be successful with Scrum you must break with the habit to sign off a detailed requirements document in advance!

We'll call it
“agile CMMI”



Conclusion: what you can take and adapt

- Some **agile principles** for requirements management: adapt them and stick to them
- An **agile requirements model**
- **Techniques to encourage the formulation of the purpose**
- **Techniques to add details to Stories** (when necessary)
- A **mapping** of the agile requirements model with regulated systems



References

Scrum and agile reading list

- *Agile Project Management with Scrum* by Ken Schwaber (Microsoft Press 2004)
- *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle (Prentice Hall 2002)
- *User Stories Applied for Agile Software Development* by Mike Cohn
- *Implementing Lean Software Development* by Mary and Tom Poppendieck (Addison Wesley 2006)
- *Agile Enterprise Requirements Information Model – Subset for Agile Project Teams- 2008* by Dean Leffingwell
www.scalingsoftwareagility.wordpress.com
- *Implementing Lean Software Development* (Addison Wesley 2006) by Mary and Tom Poppendieck
- *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage* - by Tom Gilb, Aug.2005
www.gilb.com