The software requirements document

- The software requirements document is the **official statement** of what is required of the system developers.
- Should include both a definition of **user requirements** and a specification of the **system requirements**.
- It is NOT a design document.
  - As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

Requirements document variability

- Information in requirements document **depends on type** of system and the approach to development used.
  - For example, systems developed **incrementally** will, typically, have **less detail** in the requirements document.
- Requirements documents standards have been designed e.g. IEEE standard. These are mostly applicable to the requirements for large systems engineering projects.

The structure of a requirements document

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

The structure of a requirements document

Chapter	Description
System requirements specification	This should describe the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Ways of writing a system requirements specification

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract

Natural language specification

- Requirements are written as natural language sentences supplemented by diagrams and tables.
- Used for writing requirements because it is *expressive, intuitive and universal*.
- This means that the requirements can be understood by *users and customers*.

Guidelines for writing requirements

- Invent a *standard format* and use it for all requirements.
- Use language in a consistent way. Use **shall** for mandatory requirements, **should** for desirable requirements.
- Use *text highlighting* to identify key parts of the requirement.
- Avoid the use of *computer jargon*.
- Include an *explanation (rationale)* of why a requirement is necessary.

Requirements writing style

- Do not use vague terms or verbs like “some,” “obviously,” “usually,” “often,” “it follows that,” …
- Make sure that uncompleted lists are understood completely (e.g. “etc.,” “and so on,”“…,” ...)
- Make sure that ranges are clearly understood, e.g. what means “in the range of 1 to 100”
- Ask for clear definitions of terms like “always,”“never,” “almost,” etc.
- Use pictures and examples to aid in understanding
- Explain all of your terminology
- Use “shall,” “must,” “should,” consistently

Problems with natural language

- Lack of clarity
  - Precision is difficult without making the document difficult to read.
- Requirements confusion
  - Functional and non-functional requirements tend to be mixed-up.
  - Several different requirements may be expressed together.

Example requirements for the insulin pump software system

3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. *(Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)*

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. *(A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.)*

Structured specifications

- An approach to writing requirements where the *freedom of the requirements writer is limited* and requirements are written in a standard way.
- This works well for some types of requirements e.g. requirements for embedded control system but is sometimes too rigid for writing business system requirements.

Form-based specifications

- Definition of the function or entity.
- Description of inputs and where they come from.
- Description of outputs and where they go to.
- Information about the information needed for the computation and other entities used.
- Description of the action to be taken.
- Pre and post conditions (if appropriate).
- The side effects (if any) of the function.

A structured specification of a requirement for an insulin pump

Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: safe sugar level.

Description

Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

Inputs Current sugar reading (r2); the previous two readings (r0 and r1).

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose—the dose in insulin to be delivered.

Destination Main control loop.

A structured specification of a requirement for an insulin pump

Action

CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.

Requirements

Two previous readings so that the rate of change of sugar level can be computed.

Pre-condition

The insulin reservoir contains at least the maximum allowed single dose of insulin.

Post-condition r0 is replaced by r1 then r1 is replaced by r2.

Side effects None.

Tabular specification

- Used to *supplement* natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.
- For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios.

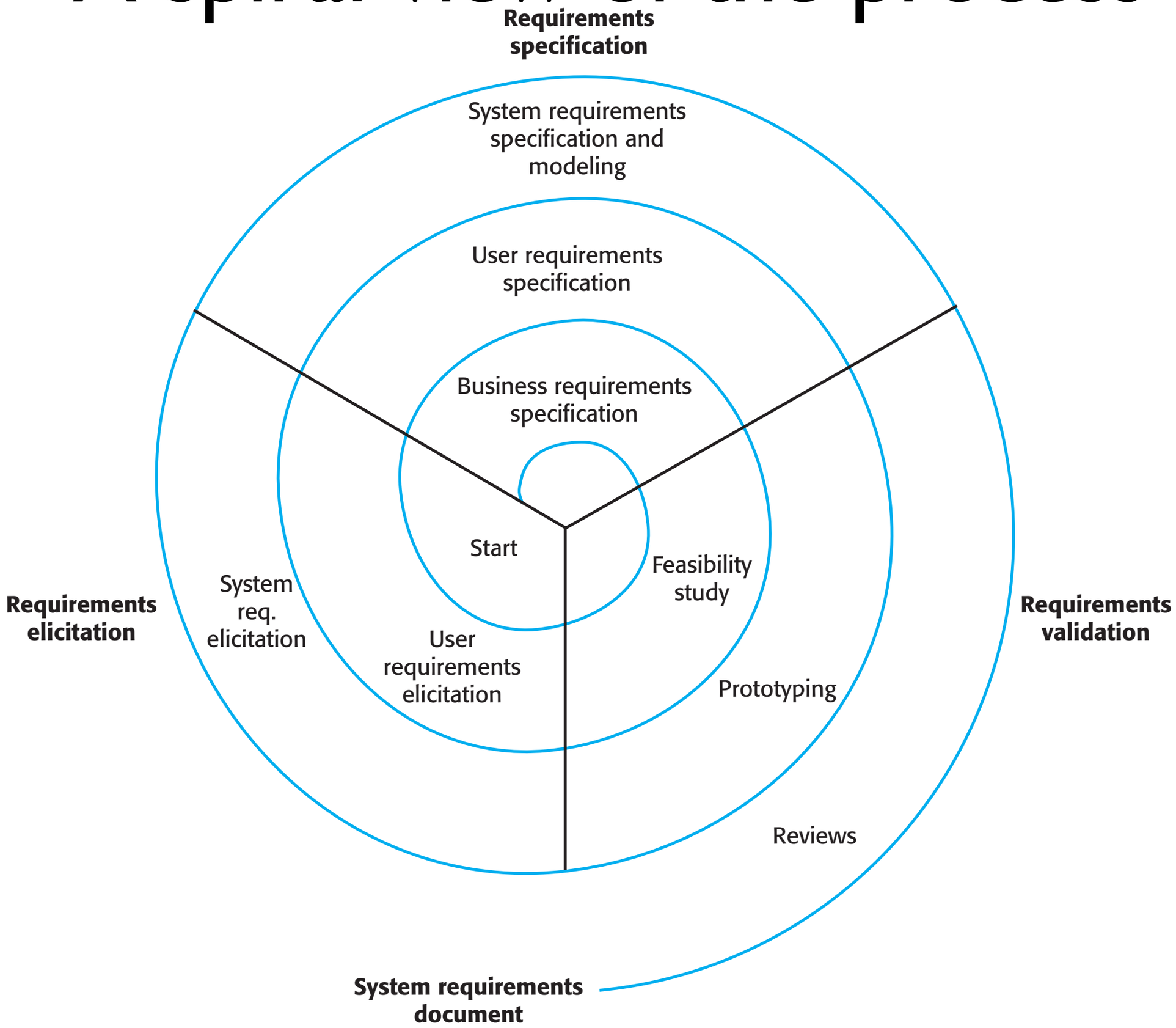
Tabular specification of computation for an insulin pump

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2 – r1) < (r1 – r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ((r2 – r1) ≥ (r1 – r0))	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

Requirements engineering processes

- Generic activities common to all processes
  - Requirements elicitation (discovery);
  - Requirements analysis;
  - Requirements validation;
  - Requirements management

A spiral view of the process



Requirements checking (validation)

- **Consistency:** Are there any requirements conflicts?
- **Completeness:** Are all functions required by the customer included?
- **Realism:** Can the requirements be implemented given available budget and technology
- **Validity:** Does the system provide the functions which best support the customer's needs?
- **Verifiability:** Can the requirements be checked?

Topics covered

- Functional and non-functional requirements
- The software requirements document
- Requirements specification
- Requirements engineering processes
- Requirements elicitation and analysis
- Requirements validation
- Requirements management

Agile methods and requirements

- Many agile methods argue that producing a requirements document is a waste of time as requirements change so quickly.
  - The document is therefore always out of date.
- Methods such as XP use incremental requirements engineering and express requirements as '**user stories**' (discussed later).

Scrum requirements?

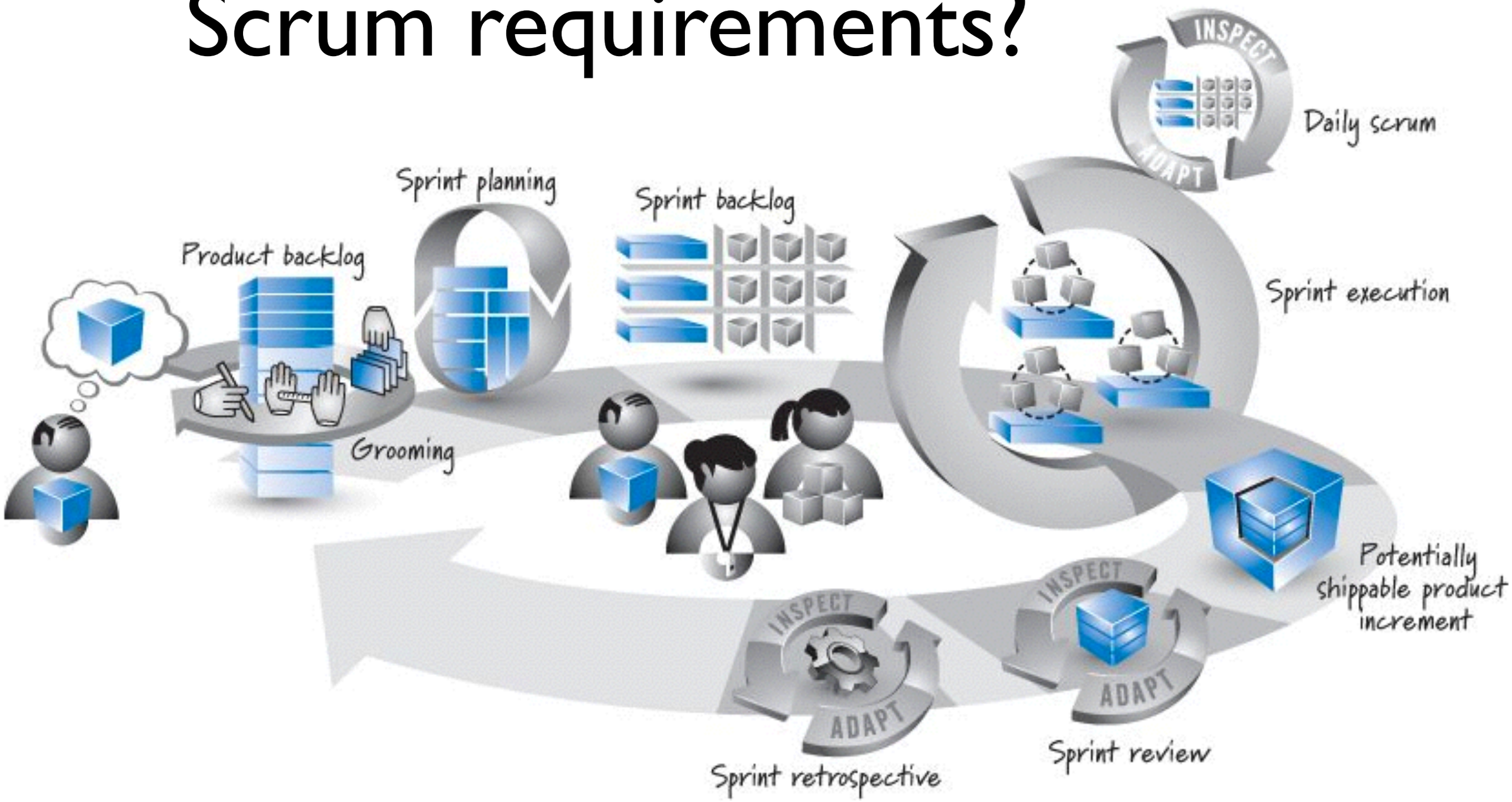
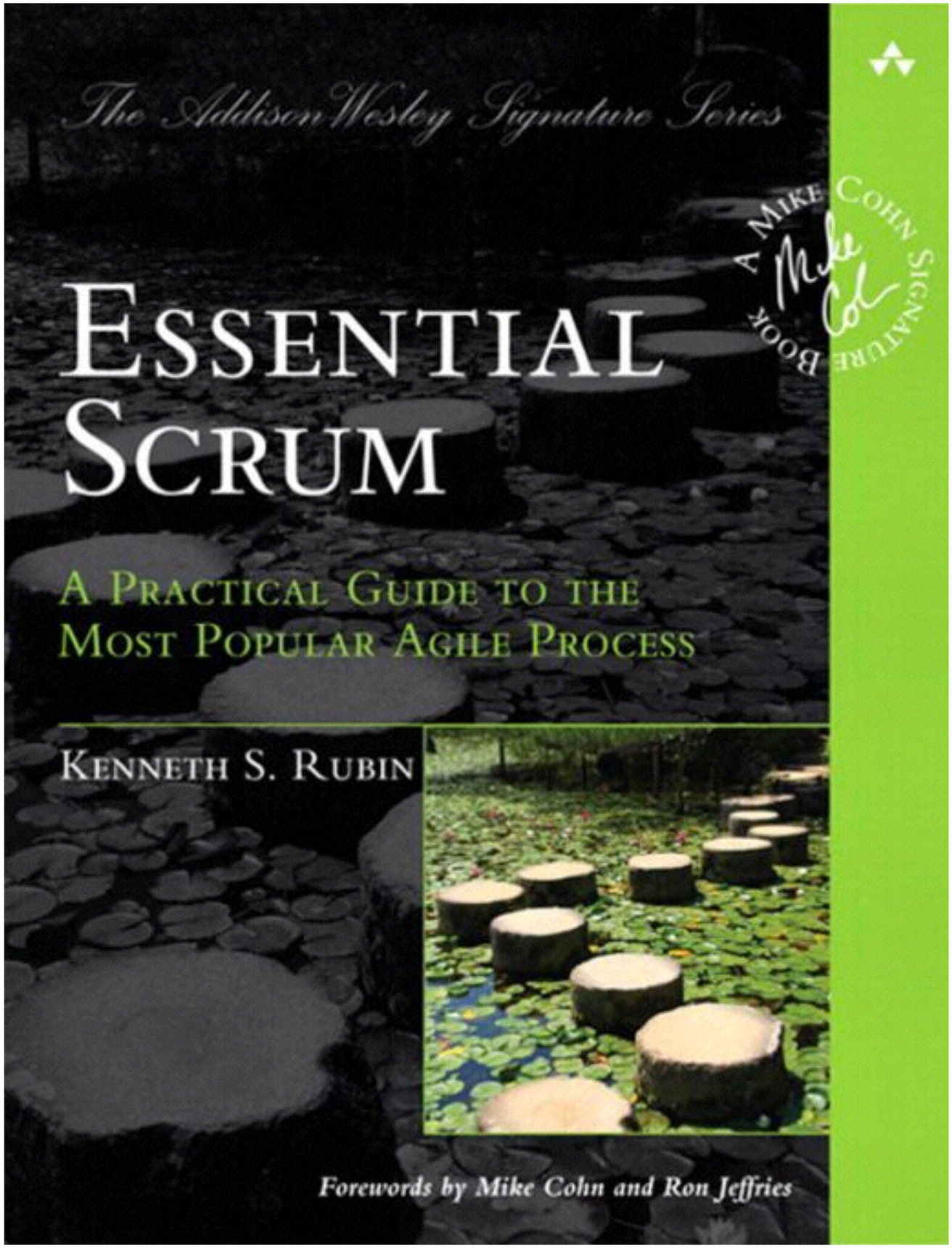


Figure 2.3. Scrum framework

Rubin, Kenneth S. Essential Scrum: A Practical Guide to the Most Popular Agile Process (Addison-Wesley Signature Series (Cohn))


The Addison Wesley Signature Series
A Mike Cohn Signature Book
ESSENTIAL SCRUM
A Practical Guide to the Most Popular Agile Process
Kenneth S. Rubin
Forewords by Mike Cohn and Ron Jeffries

User Story

- Ron Jeffries offers a simple yet effective way to think about user stories (Jeffries 2001).
- He describes them as the three Cs
  - Card
  - Conversation
  - Confirmation

Rubin, Kenneth S. Essential Scrum: A Practical Guide to the Most Popular Agile Process (Addison-Wesley Signature Series (Cohn))

User Story - Cards

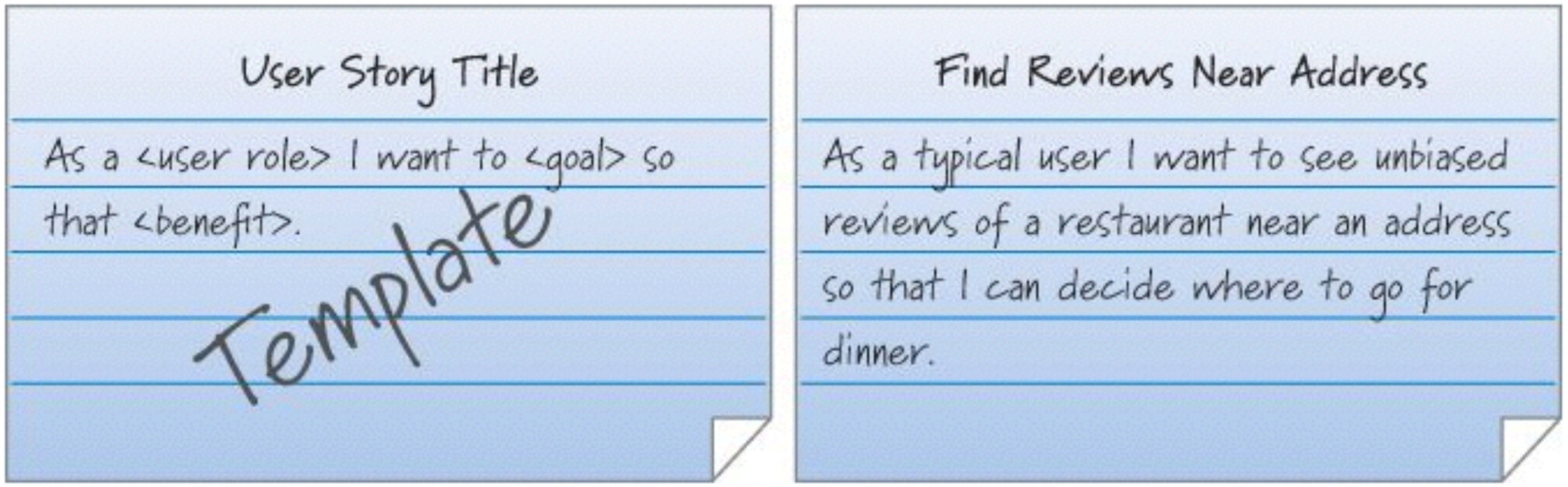


Figure 5.2. A user story template and card

Rubin, Kenneth S. Essential Scrum: A Practical Guide to the Most Popular Agile Process (Addison-Wesley Signature Series (Cohn))

User Story - Conversation

- The details of a requirement are *exposed and communicated* in a conversation among the development team, product owner, and stakeholders.
- The user story is simply *a promise (starting point)* to have that conversation.

Rubin, Kenneth S. Essential Scrum: A Practical Guide to the Most Popular Agile Process (Addison-Wesley Signature Series (Cohn))

User Story - Conversation

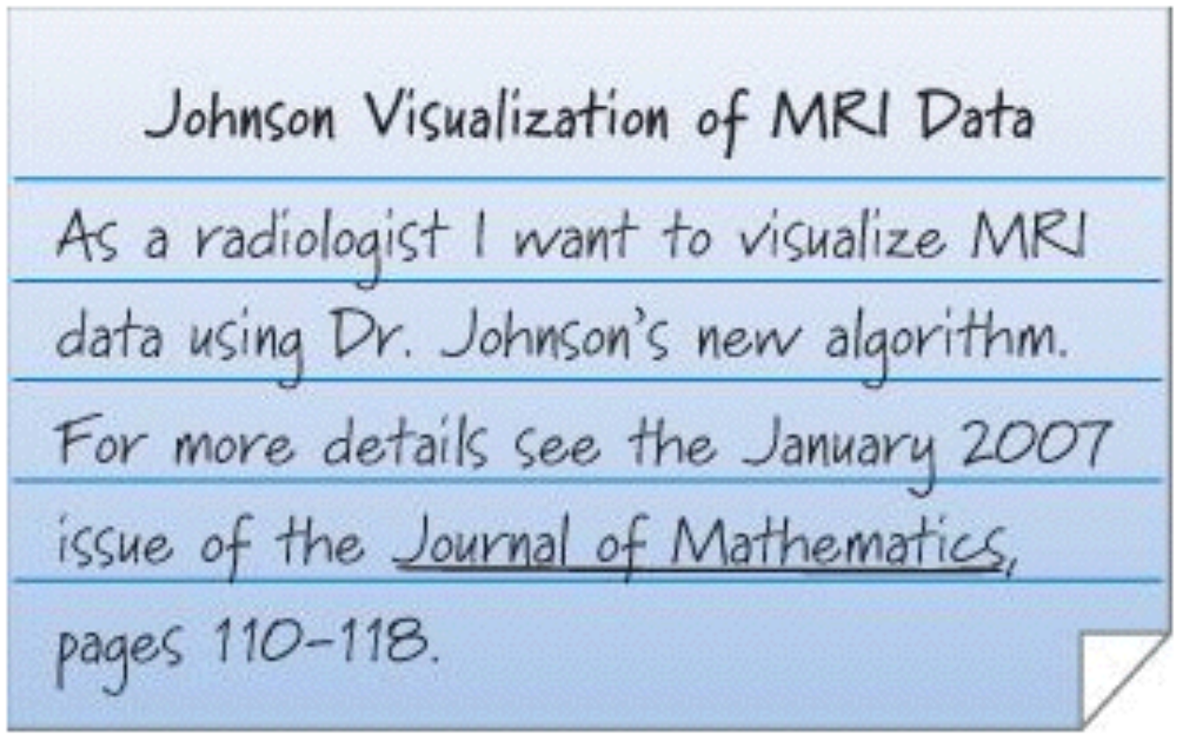


Figure 5.3. User story with additional data attached

Rubin, Kenneth S. Essential Scrum: A Practical Guide to the Most Popular Agile Process (Addison-Wesley Signature Series (Cohn))

User Story - Confirmation

- A user story also contains *confirmation information* in the form of conditions of satisfaction.
- These are *acceptance criteria* that clarify the desired behavior.
- They are used by the development team to *better understand what to build* and *test* to confirm that the user story has been implemented to their satisfaction.

Rubin, Kenneth S. Essential Scrum: A Practical Guide to the Most Popular Agile Process (Addison-Wesley Signature Series (Cohn))

User Story - Confirmation

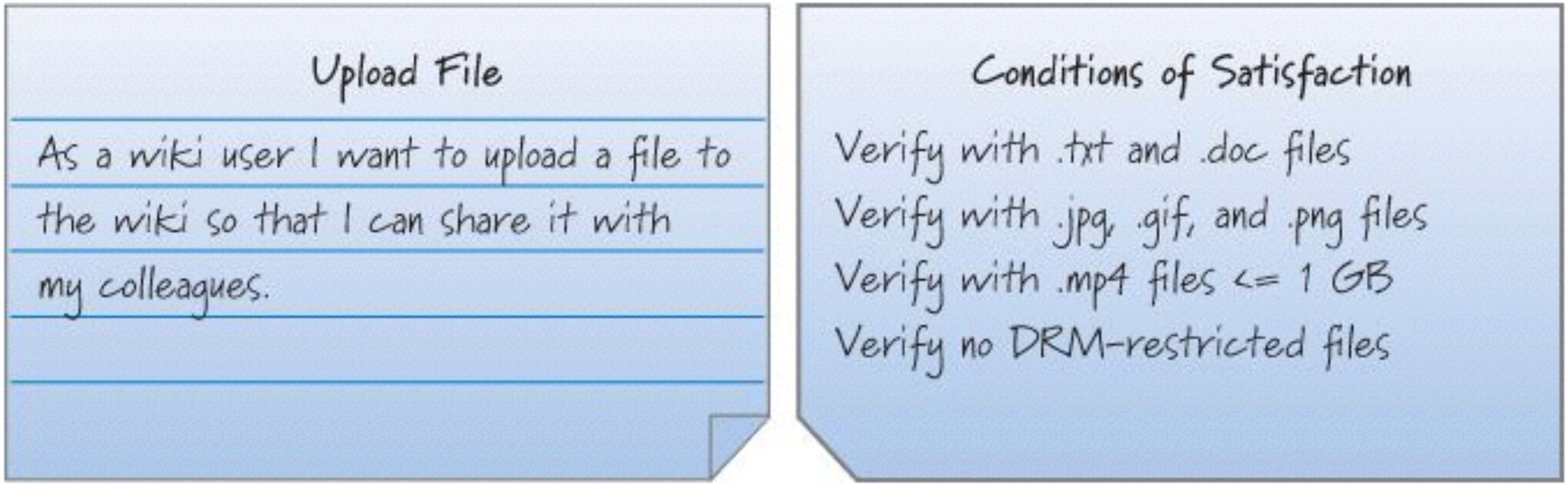


Figure 5.4. User story conditions of satisfaction

INVEST in good stories

- Independent
- Negotiable
- Valuable
- Estimable
- Small (Size appropriately)
- Testable

Story map

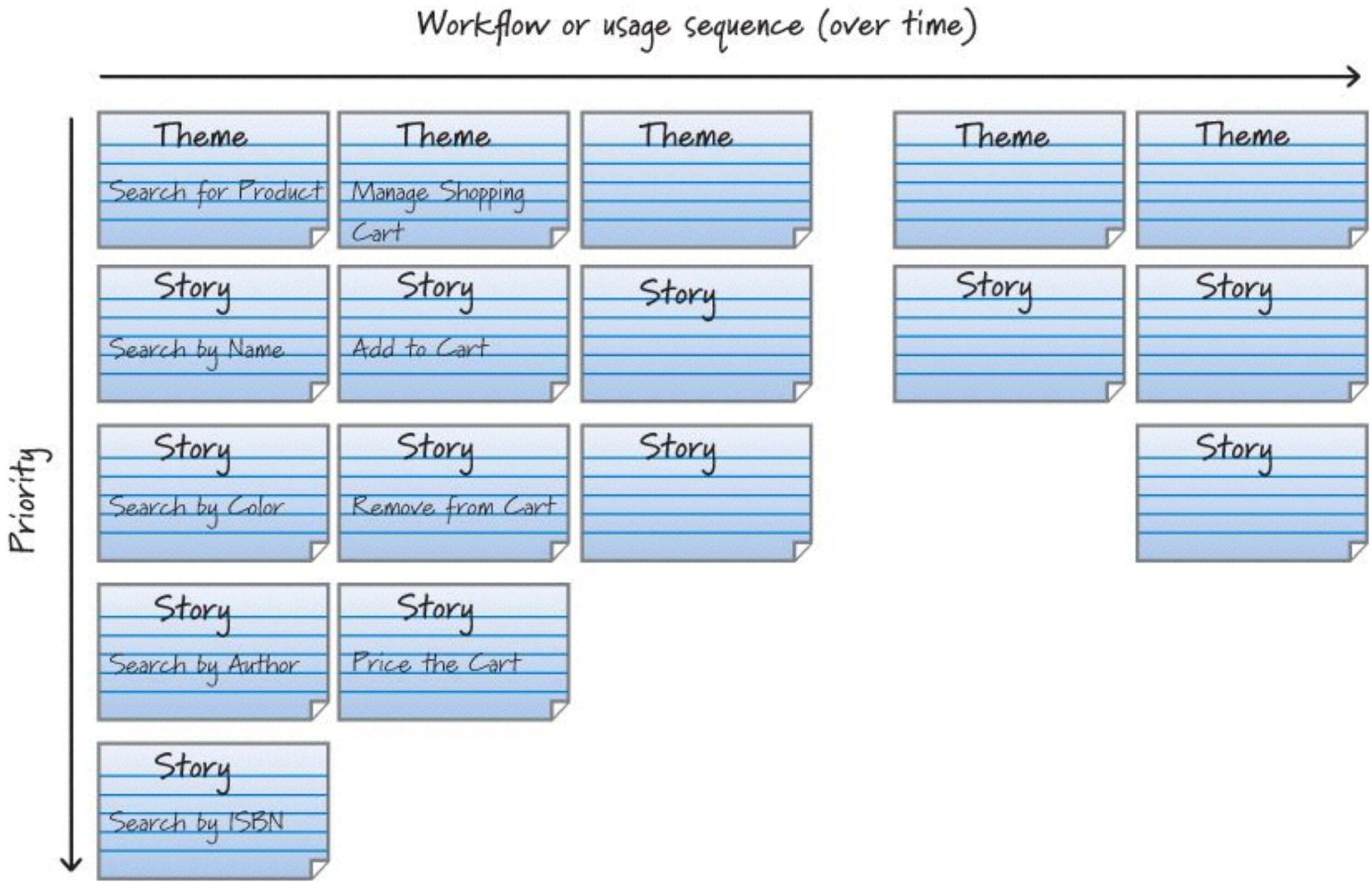
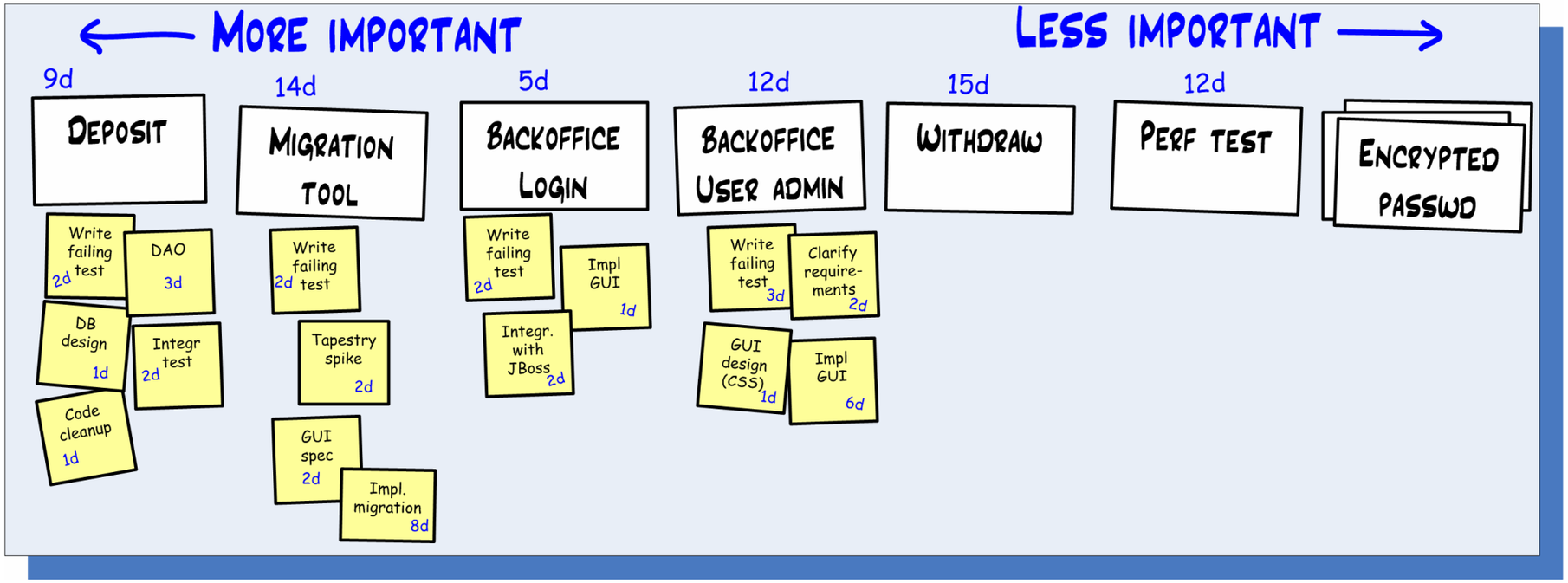


Figure 5.13. Story map

Sprint planning



http://infoq.com/minibooks/ scrum-xp-from- the-trenches

Product/sprint backlog example (March 3)

sprint id	user story (task +description)	Est (hours)	assigned to
…	…	…	…
…	…	…	…
…	…	…	…

Product Backlog

	Item #	Description	Est	By
Very High				
	1	**Finish database versioning**	16	KH
	2	**Get rid of unneeded shared Java in database**	8	KH
	-	**Add licensing**	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	**Enforce unique names**	-	-
	7	In main application	24	KH
	8	In import	24	AM
	-	**Admin Program**	-	-
	9	Delete users	4	JM
	-	**Analysis Manager**	-	-
	10	When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	-	**Query**	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
	-	**Population Genetics**	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M

sample from Eclipse.org

Sprint Backlog

Requirement	Task	Who's working	Status	Work Left [Hours]				
				Day 1	Day 2	Day 3	Day 4	Day 5
Member Sign In	Database Coding	Anjali	Done	1	0	0		
	Unit Testing Scripts	Anjali	Done	2	0	0		
	Business Logic	Anjali	Done	1	0	0		
	UI Test Scripts	Anu	Done	2	2	0		
	Front End Screens	Anjali	Done	2	2	0		
	Load Testing	Anu/ Anjali	Done	2	2	1		
Reset Password	Unit Testing Scripts	Thomas	Done	1	0	0		
	Business Logic	Thomas	Done	1	0	0		
	UI Test Scripts	Anu	Done	1	1	0		
	Front End Screens	Thomas	Done	1	1	1		
	Integration Testing	Anu	Pending	1	1	1		
Change Password	Unit Testing Scripts	Thomas	Done	0.5	0	0		
	Business Logic	Thomas	Done	0.5	0	0		
	UI Test Scripts	Anu	Done	0.5	0	0		
	Front End Screens	Thomas	Done	0.5	0	0		
	Integration Testing	Anu	Pending	0.5	0.5	1		
Change Email	Unit Testing Scripts	Anjali	Done	0.5	0	0		
	Business Logic	Anjali	Done	0.5	0	0		
	UI Test Scripts	Anu	Done	0.5	0	0		
	Front End Screens	Anjali	Done	0.5	0	0		
	Integration Testing	Anu	Pending	0.5	0.5	0.5		
Help Link	Front End Screens	Anjali	Pending	0.5	0.5	0.5		
	Manual Testing	Anu	Pending	0.5	0.5	0.5		
	Integration Testing	Anu	Pending			0.5		
			Work Remaining	21	11	6		

Jurgen Appelo – jurgen@noop.nl

Sprint backlog example

Table 20.1. Sprint Backlog with Estimated Effort Remaining Each Day

Tasks	D1	D2	D3	D4	D5	D6	D7	D8	D9	. . .	D15
Task 1	8	4	4	2							
Task 2	12	8	16	14	9	6	2				
Task 3	5	5	3	3	1						
Task 4	7	7	7	5	10	6	3	1			
Task 5	3	3	3	3	3	3	3				
Task 6	14	14	14	14	14	14	14	8	4		
Task 7						8	6	4	2		
Tasks 8–30	151	139	143	134	118	99	89	101	84		0
Total	200	180	190	175	155	130	115	113	90		0

Rubin, Kenneth S. Essential Scrum: A Practical Guide to the Most Popular Agile Process (Addison-Wesley Signature Series (Cohn))

Burn-down Chart

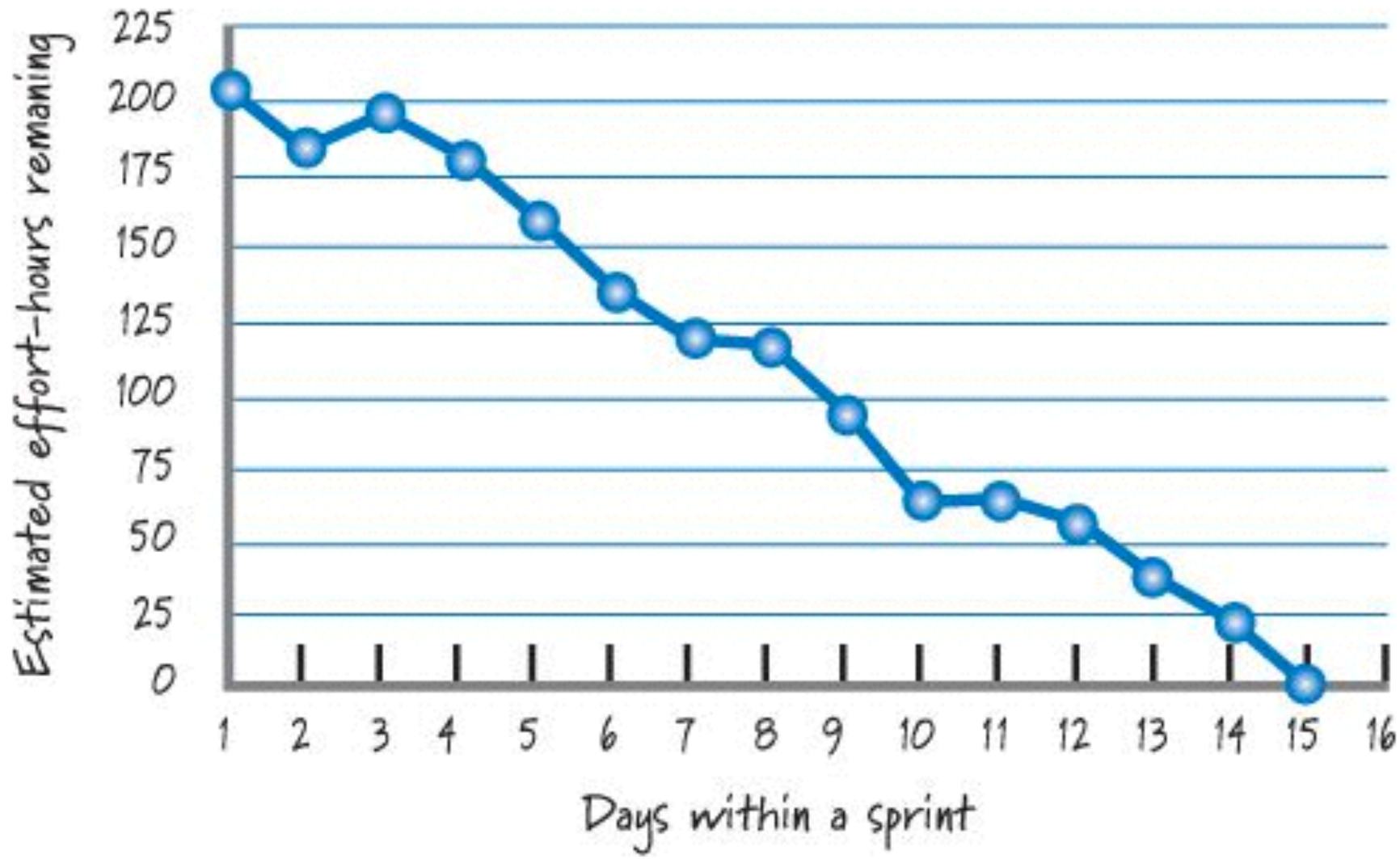


Figure 20.7. Sprint burndown chart

Rubin, Kenneth S. Essential Scrum: A Practical Guide to the Most Popular Agile Process (Addison-Wesley Signature Series (Cohn))