IN5140 – Smart processes and agile methods in software engineering

Lecture 8 November 2022:

Measurements used in process improvement

Professor Dag Sjøberg

Structure

- Measurement theory
- Concepts and Constructs
- A study of Scrum versus Kanban

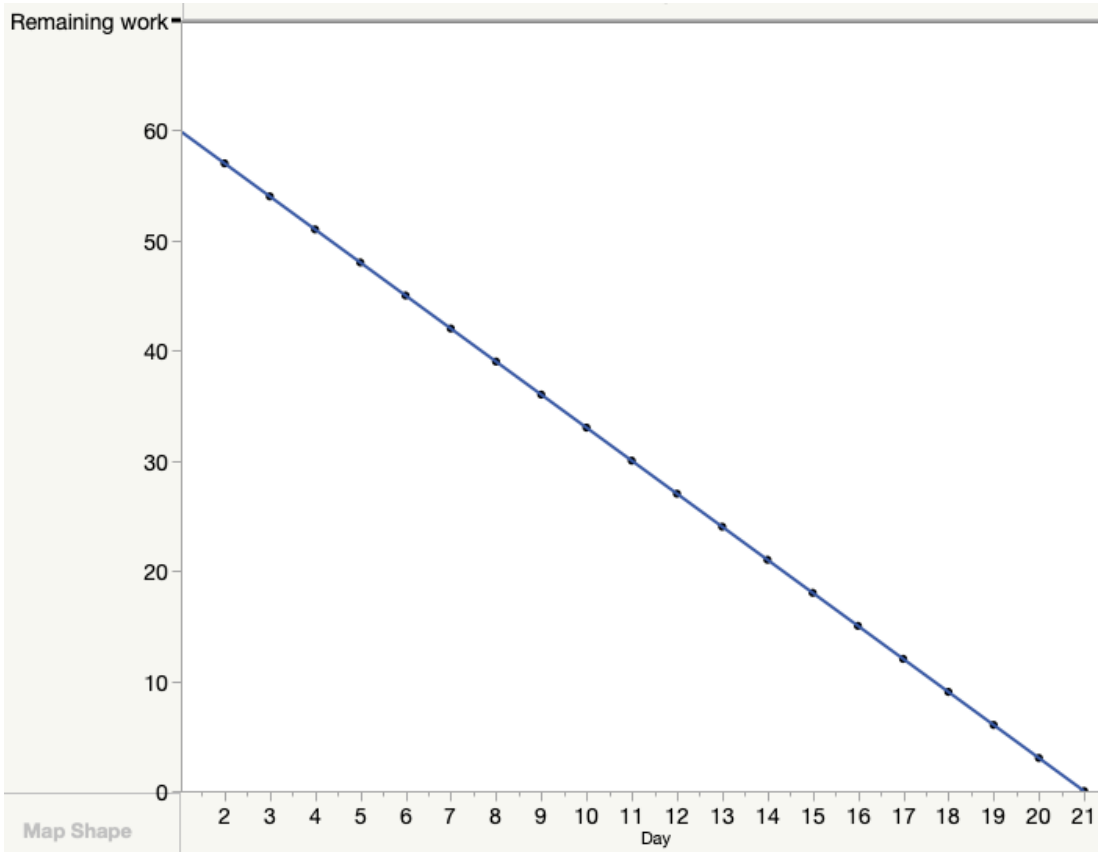
Why should we measure?

Measurements are central in all kinds of improvement work

- including software process improvement, both plan-driven and agile processes

Measurements for many purposes

- Basis for cost and time estimation
- Make results or progress visible, cf. ”burn down chart”
- Feedback to experts. Research shows that «experts» who don’t receive feedback, learn little
- Thousands of other purposes …



Burn down chart, book p. 128

Knowledge about measurements is useful whatever the discipline

- An enormous amount of data in our digital era. More important than ever to know
  - how data has been produced and
  - its quality
- Can you trust the data, how reliable is it?
  - fake news
  - echo chambers
  - conspiracy theories
  - basis for political decisions
  - basis for research

Measurements are relevant to your project:

- “Based on the improvement goal(s) in your project, identify and describe a minimum of three measures to be used to assess the effects of process changes. For each measure, describe:
  - Who will collect/report data?
  - When (how often) will data be corrected?
  - How is data collected. For example, which tools are used?
  - How is data quality and validity ensured. For example, who is responsible?
- Also, discuss possible challenges related to data collection and validity. Note that you do not have to collect all data for all the measures if practically difficult.”

Said about measurement

To measure is to know.
If you cannot measure it, you cannot improve it.
Lord Kelvin

Not everything that counts can be measured.
Not everything that can be measured counts.
Albert Einstein

In God we trust, all others bring data –
W. Edwards Deming


William Thomson, Lord Kelvin.

Quantitative data

- Data expresses quantity
- Data expressed as numbers
- Used in statistics

Qualitative data

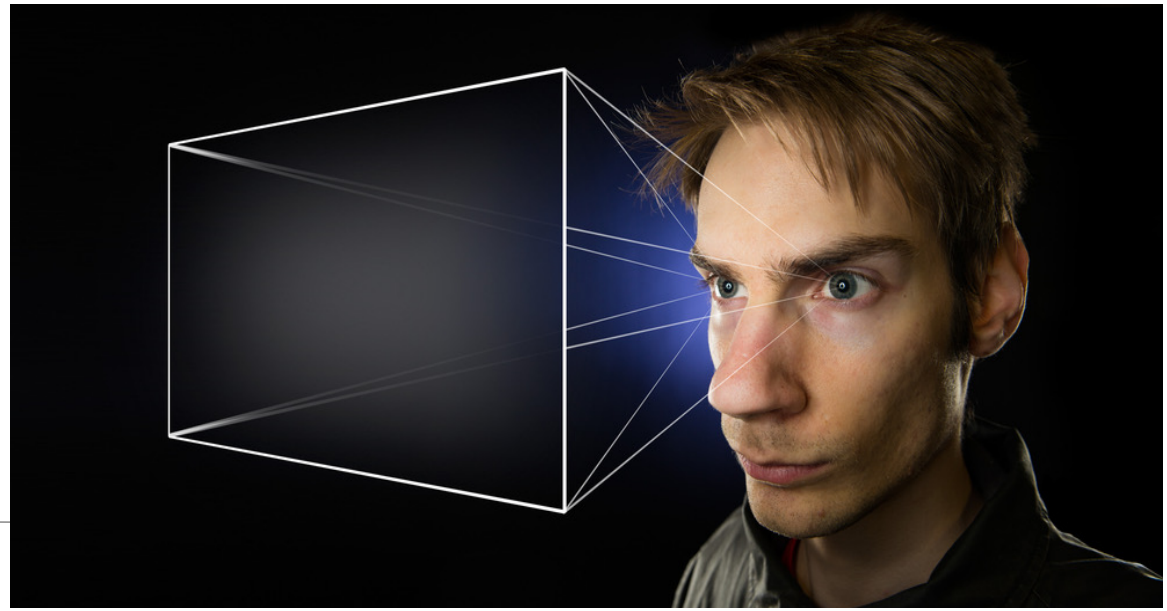
- Data expresses quality in some sense
- Data expressed as text, images, audio or video but not numbers
- Not used in statistics

Objective data

- Based on facts rather than feelings, opinions, prejudices or interpretations [Merriam-Webster]

Subjective data

- Related to the way people experience things in their own mind
- Based on feelings or opinions rather than facts, modified or affected by personal views, experience or background [Merriam-Webster]



Objective vs. subjective data

- We usually prefer objective data
- However, good, subjective data on something relevant is more important than objective data on irrelevant aspects
- When introducing measurements in an organization, data will often be subjective to begin with. Later, we may be able to measure more aspects objectively by using better methods for data collection

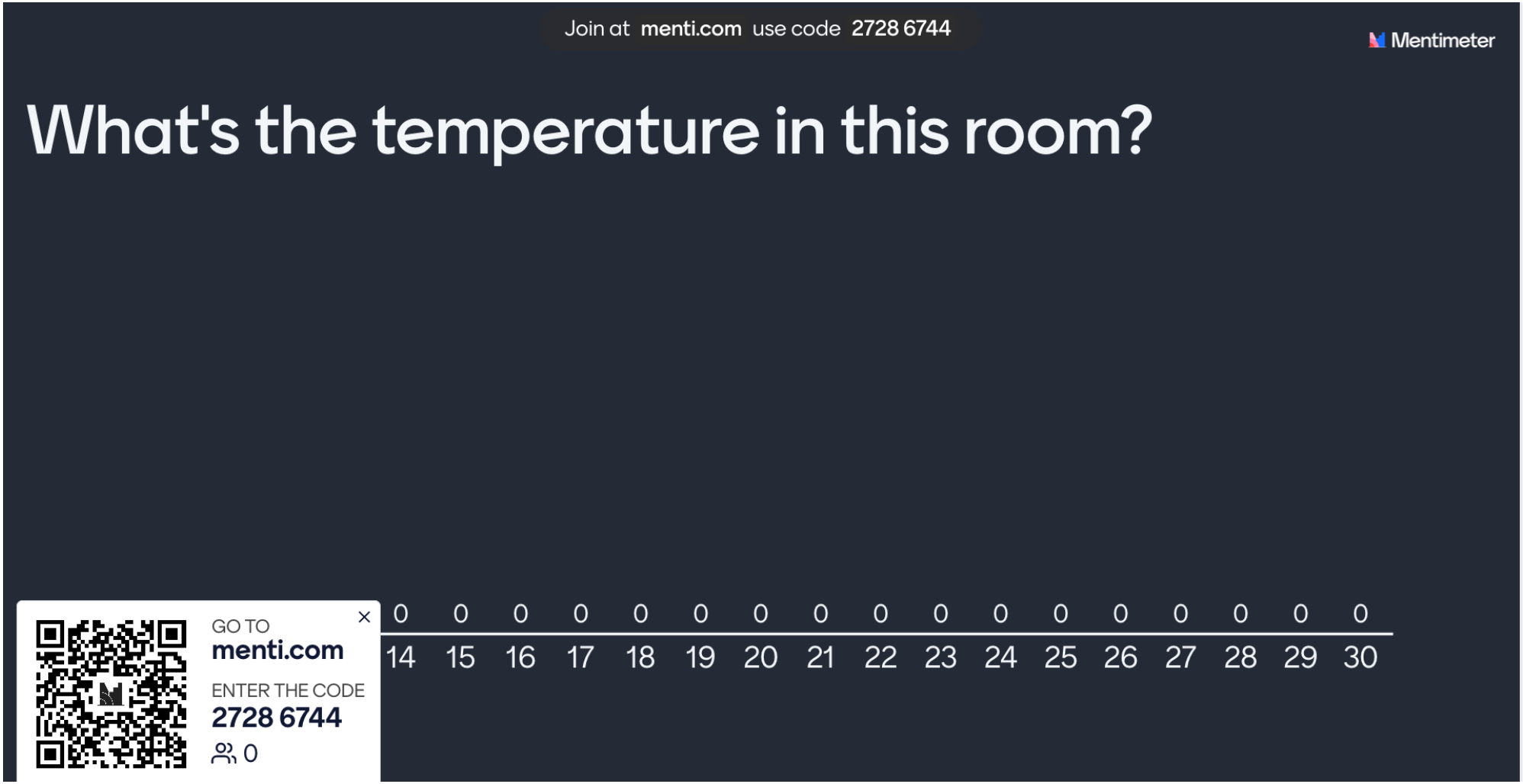
Objective measurement

- Usually, the measurement process can be automated
- (Almost) no random measurement error, i.e., the process is perfectly reliable
- However, imprecise definitions may cause different people to measure the phenomenon differently and thus obtain different results (see *construct validity* later)

Subjective measurement

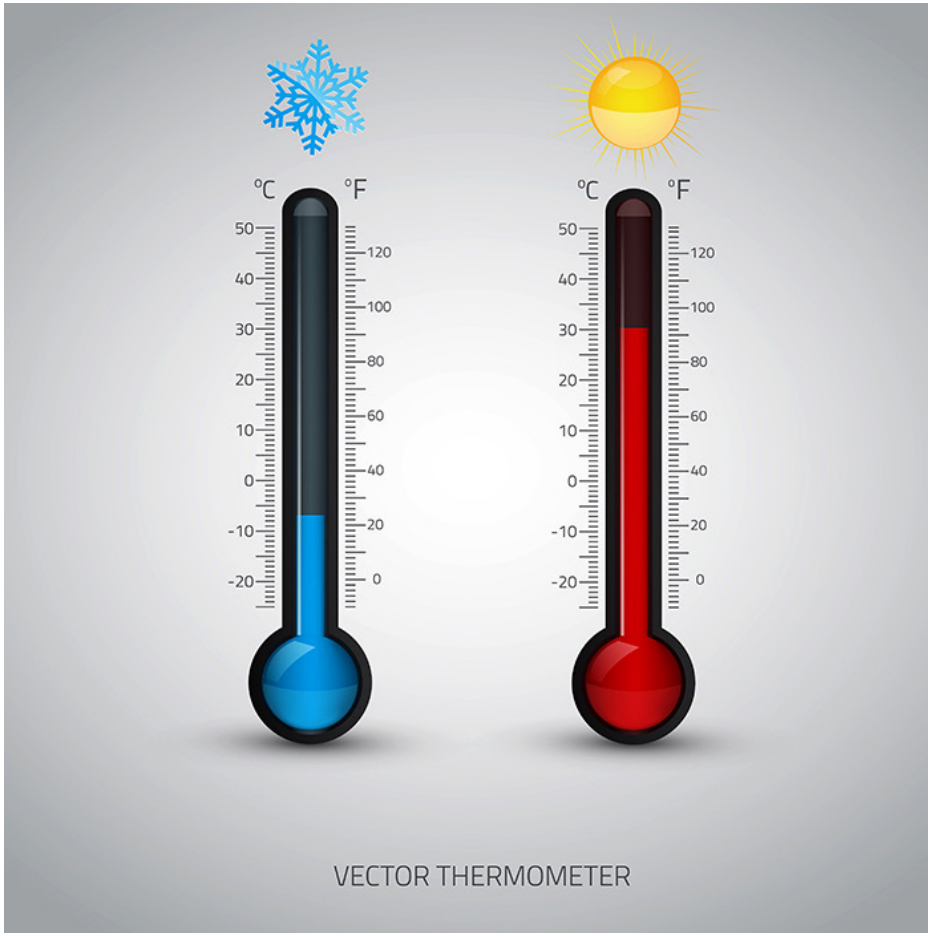
- Human involvement in the measurement process
- If we repeat the measurement of the same object(s) several times, we might not get exactly the same measured value every time, i.e., the measurement process is not perfectly reliable

Mentimeter


Join at menti.com use code 2728 6744
Mentimeter
What's the temperature in this room?
GO TO
menti.com
ENTER THE CODE
2728 6744
0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

Don’t confuse objective/subjective with quantitative/qualitative

- Although objective data is often quantitative and subjective qualitative, objective data may be qualitative and subjective quantitative



Types of measurement scale

Scale Type	Characterization	Examples (generic)	Examples (software engineering)
Nominal	**Divides the set of objects into categories, with no particular ordering among them**	**Labeling, classification**	**Name of process model** **Defect type**
Ordinal	**Divides the set of entities into categories that are ordered**	**Preference, ranking, difficulty, Likert scales**	**Failure severity** **Complexity of software**
Interval	**Comparing the differences between values is meaningful**	**Calendar time, temperature (Fahrenheit, Reaumur, Celsius)**	**Start and end date of activities**
Ratio	**There is a meaningful "zero" value, and ratios between values are meaningful**	**Length, weight, time intervals, absolute temperature (Kelvin)**	**Lines of code** **Lead time** **Number of errors** **Cost per function**

Operations

Scale	Basic Empirical Operations	Mathematical Group Structure	Permissible Statistics (invariantive)
NOMINAL	Determinaflon of equality	*Permutation group* $x' = f(x)$ $f(x)$ means any one-to-one substitution	Number of cases Mode Contingency correlation
ORDINAL	Determination of greater or less	*Isotonic group* $x' = f(x)$ $f(x)$ means any monotonic increasing function	Median Percentiles
INTERVAL	Determination of equality of intervals or differences	*General linear group* $x' = ax + b$	Mean Standard deviation Rank-order correlation Product-moment correlation
RATIO	Determination of equality of ratios	*Similarity group* $x' = ax$	Coefficient of variation

Mode: the value that appears most often

On the Theory of *Scales* of *Measurement*. S. S. *Stevens*. Science, New Series, Vol. 103, No. 2684. (Jun. 7, *1946*), pp. 677-680

Quantitative. Required for "normal" measurement

The mathematics/statistics is not syllabus but good to know

Included by Gunnar Bergersen 25 October:

Likert type scales

- **Evaluation-type**

  Example:

  – **"Familiarity with and comprehension of the software development environment"**

  - ❑ Little
  - ❑ Unsatisfactory
  - ❑ Neutral
  - ❑ Satisfactory
  - ❑ Excellent

- **Frequency-type**

  Example:

  –**"Customers provide information to the project team about the requirements"**

  - ❑ Never
  - ❑ Rarely
  - ❑ Neutral
  - ❑ Occasionally
  - ❑ Most of the time

- **Agreement-type**

  Example:

  –**"The tasks supported by the software at the customer site change frequently"**

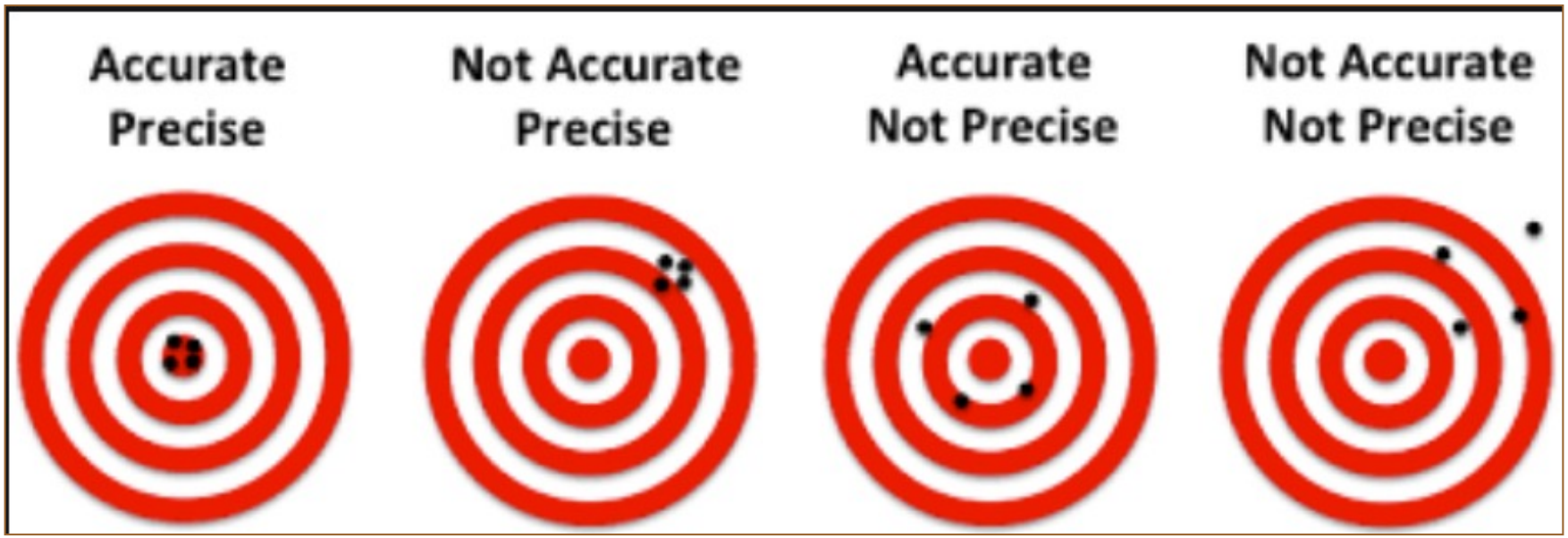
  - ❑ Strongly Agree
  - ❑ Agree
  - ❑ Neutral
  - ❑ Disagree
  - ❑ Strongly Disagree

Data should be validated

- Check whether single and aggregated data are reasonable
- Are values outside what was expected, then you should identify the reasons. Is it due to special incidents or error in the data collection?

Accuracy and precision of data

- Accuracy: how close a measured value is to the actual (true) value
- Precision: how close the measured values are to each other
  - How many digits are used?
  - Which measurement scale is used?



Structure

- Measurement theory
- Concepts and Constructs
- A study of Scrum versus Kanban

Research or doing a Master's thesis is about improving something

You would often like to demonstrate that something (A) is better than something else (B)

How to know whether research or your thesis will lead to improvement?

How to show that A is better than B?

- Ideally, demonstrate improvement by in relevant success criteria using a validated measurement instrument

Claims: "A method leads to faster development"

Measurement tool:

"A smart technology reduces energy consumption"

Measurement tool:

However, many success criteria and other relevant aspects are not directly measurable

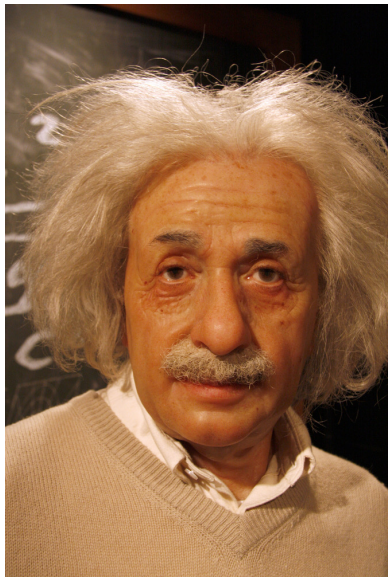
- Functionality
- Quality
- Skill
- …

Concept

Concepts are fundamental in development and acquisition of knowledge

- Concepts categorize and generalize over particulars and abstract over details
- Concepts organize complex notions and thus increase our overall level of knowledge

Well-defined concepts are at the core of all sciences



“thinking without the positing of categories and of concepts in general would be as impossible as is breathing in a vacuum”

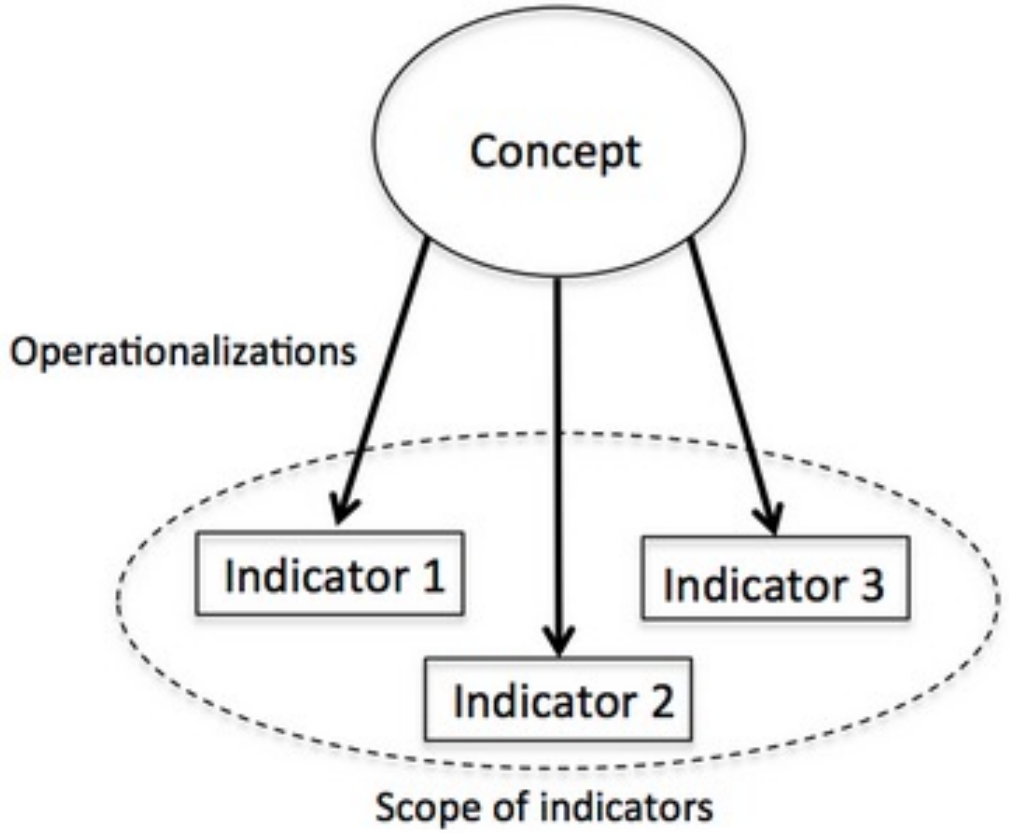
- **A precise, well-understood concept is the basis for making meaningful and comparable measures of the concept**
- **How to measure the concept will of course depend on the chosen definition**

How do we measure a concept?

- Simple concepts like *time* and *temperature* are straightforward to measure
- For comprehensive concepts that are not directly measurable one needs to define one or more *indicators*
  - How do we measure *study quality* or *software quality*?

Construct = concept + indicators

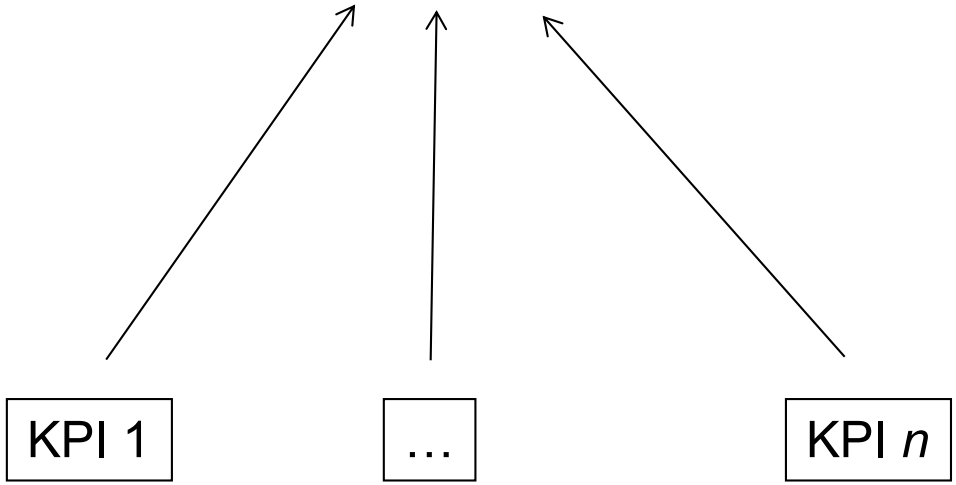
- The process of defining (measurable) indicators is called *operationalization*
- A *construct* is a concept that is operationalized into a set of indicators
- *Construct validity* = how well the measurements (indicators) represent the concept (= begreps-validitet på norsk)



D.I.K. Sjøberg and G.R. Bergersen. Construct Validity in Software Engineering, *IEEE Transactions on Software Engineering*, 2022, doi: 10.1109/TSE.2022.3176725

Conceptual level

Company performance



Operational (measurable) level

KPI (Key Performance Indicator)

A key performance indicator (KPI) is a type of performance measurement. KPIs evaluate the success of an organization or of a particular activity (such as projects, programs, products and other initiatives) in which it engages.

[Wikipedia]

Agile development KPIs

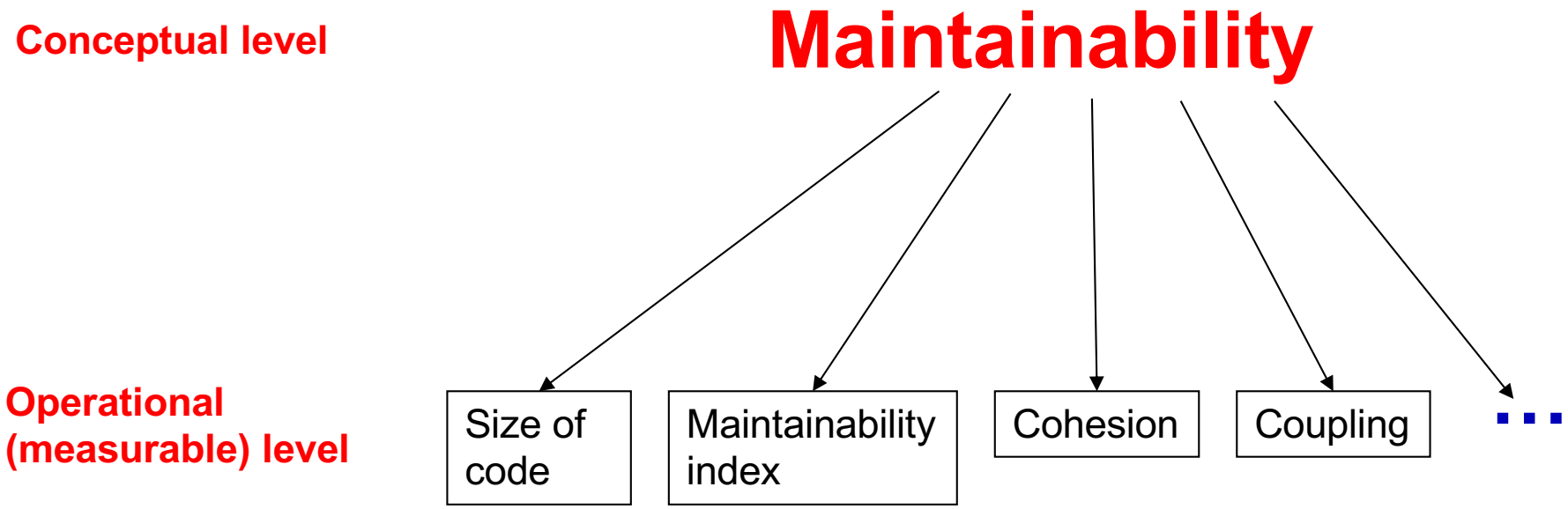
- **Capacity/throughput/velocity:**
  the number of features/user stories delivered per unit of time (in Scrum: number of features/user stories delivered per sprint)
- **Lead-time (cycle time):**
  the time it takes to finish a user story/work item
- **Code coverage by automated tests**

System quality attributes in ISO 25010

- Functional suitability
- Reliability
- Usability
- Performance efficiency
- **Maintainability**
- Portability
- Compatibility
- Security

High level concepts (e.g., quality) may be represented by subconcepts, which in turn are represented as measurable indicators

Example indicators of maintainability (how easy it is to maintain a piece of software)



Size of source code

- Lines of code *without* comment lines
- Lines of code *with* comment lines
- Number of classes (or files, methods, etc.)

Maintainability index, a formula that combines:

- Lines of code
- Cyclomatic complexity (McCabe)
- Halstead complexity measures

Cyclomatic Complexity

- –The complexity **M** is then defined as
- $M = E - N + 2$, where
- E = the number of edges of the graph. N = the number of nodes of the graph.

*Details here are not part of the syllabus

Node	Statement
(1)	`while(x<100){`
(2)	`  if (a[x] % 2 == 0) {`
(3)	`    parity = 0;`
	`  }`
	`  else {`
(4)	`    parity = 1;`
(5)	`  }`
(6)	`  switch(parity){`
	`    case 0:`
(7)	`      println( "a[" + i + "] is even");`
	`    case 1:`
(8)	`      println( "a[" + i + "] is odd");`
	`    default:`
(9)	`      println( "Unexpected error");`
	`  }`
(10)	`  x++;`
	`}`
(11)	`p = true;`

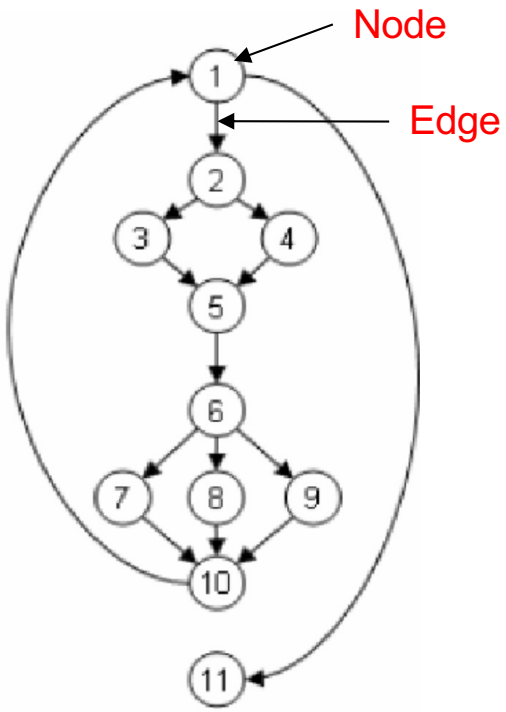
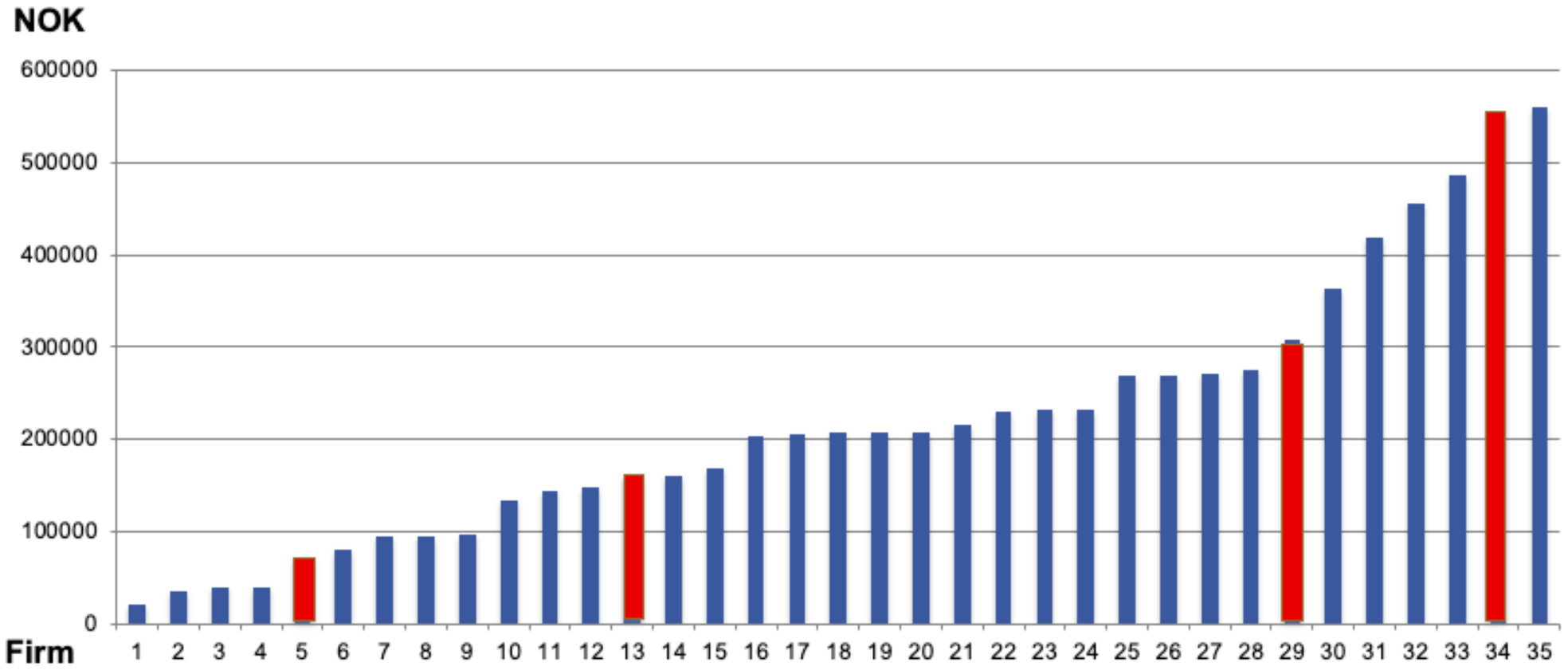


Figure from Madi, Ayman et al. "On the Improvement of Cyclomatic Complexity Metric" (2013)

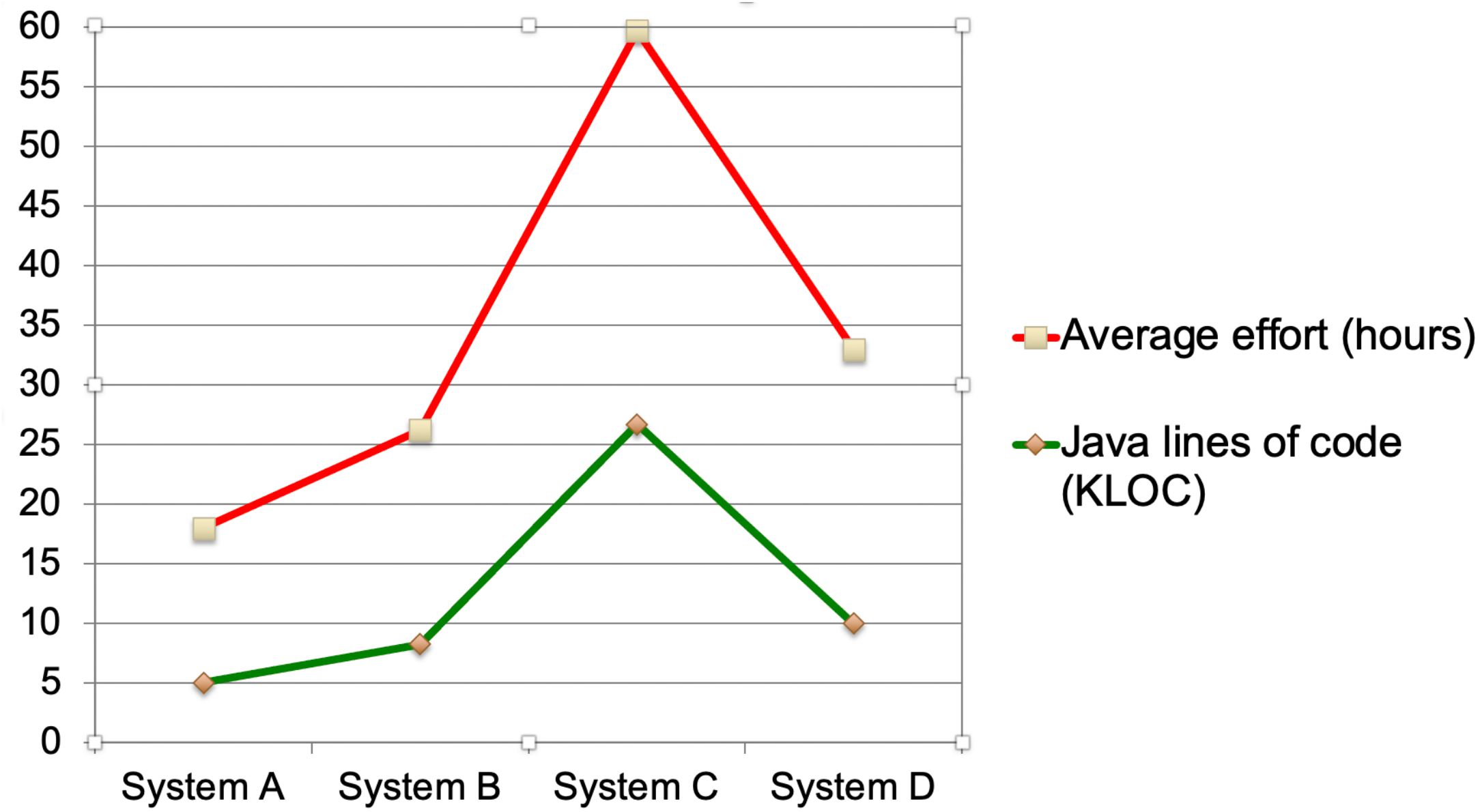
Software complexity

- "Cyclomatic complexity" is often referred to as "software complexity"
- Is "software complexity" a good term?
  - Something is *complex* if it is "not easy to analyze or understand" [The Oxford Dictionary of Difficult Words]
- Other factors that add to the complexity of software that are not captured by this formula?
  - Naming of identifiers?
  - logical structure?
- Is "software complexity" objective of subjective?
  - Complex for whom?

Four companies developed the same system

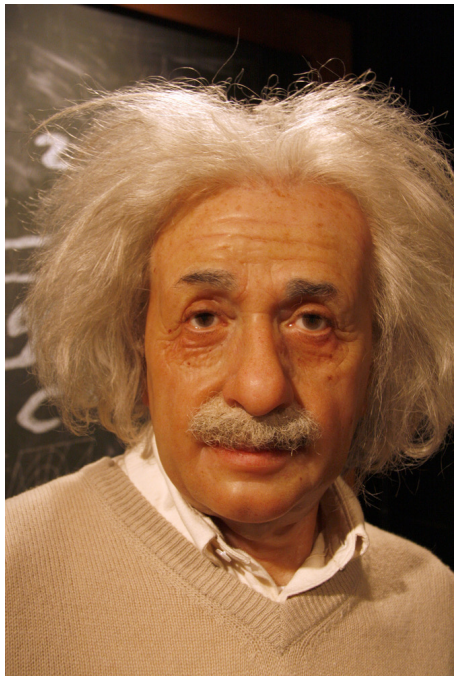


Code size versus maintainability



Consequence: “small is beautiful”

“One principle that every software developer should follow is that “small is beautiful”. Developers should therefore put some extra effort in making their systems small. Our own, controlled studies show that smaller design and code, given other factors constant, lead to more understandable and maintainable systems. Also a large number of other studies show that size correlate negatively with many quality attributes. Note that this does not imply that the smallest possible system is the best. An analogy is written text—it should be minimal, but not less than minimal, or as Einstein stated: “Everything should be made as simple as possible, but not simpler.”



[Walter Tichy: Empirical software research: an interview with Dag Sjøberg, University of Oslo, Norway.” *ACM Ubiquity.* (June 2011) http://dl.acm.org/citation.cfm?id=1998374]

See book Section 4.4.4 “Develop minimal software”

Structure

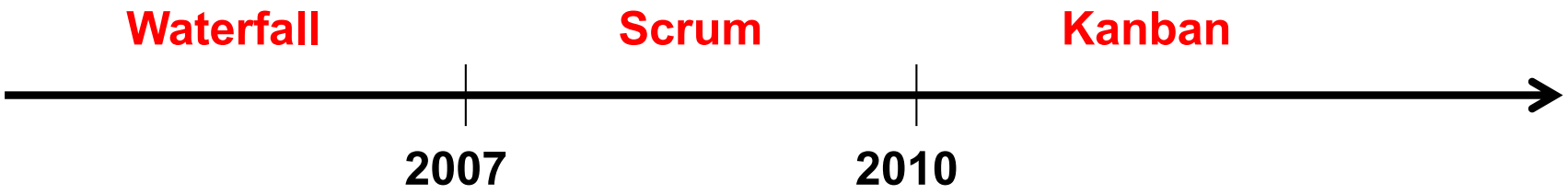
- Measurement theory
- Concepts and Constructs
- A study of Scrum versus Kanban

Software Innovation (part of Tietoevry)

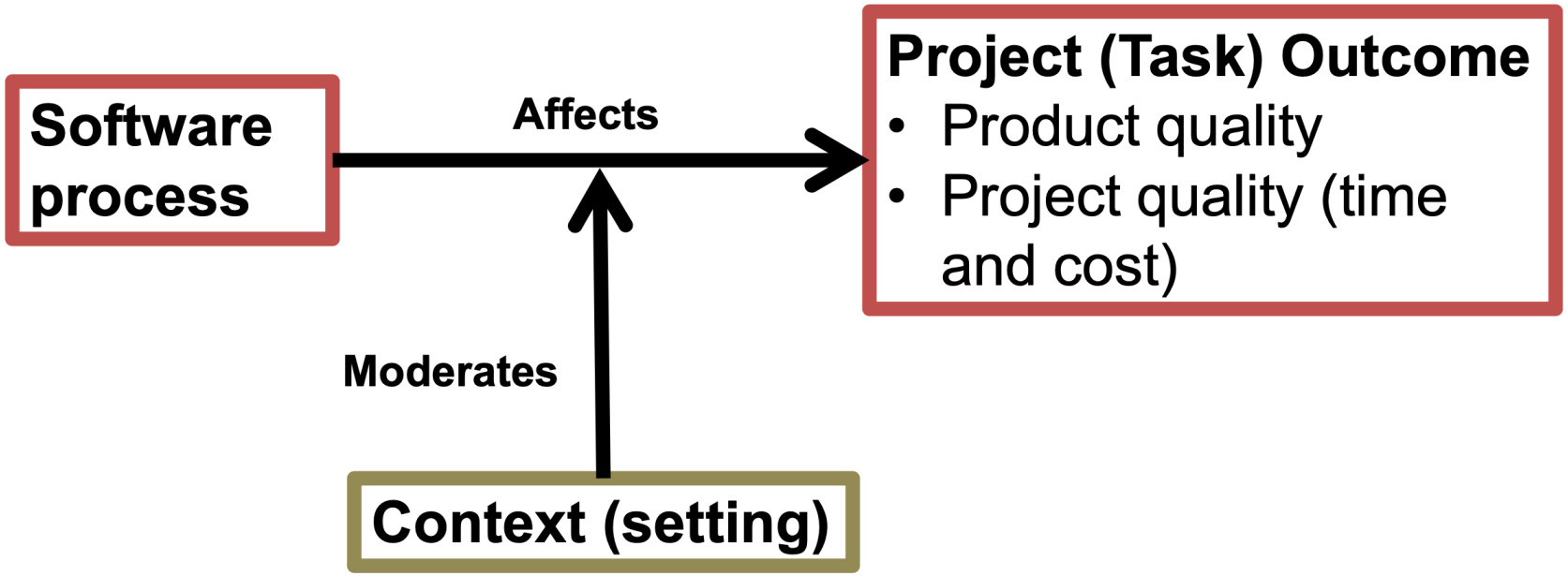
- Scandinavian software house that develops document management systems
- 350 employees, more than 400 customers
- 100 developers and specialists working document management systems
- 10 development teams

Study of Scrum versus Kanban

- Changed to Kanban in 2010
- Where the claimed benefits of Kanban met?
- Had production, and project and product quality improved?
- A study to find out was run as a research collaboration between University of Oslo and Software Innovation



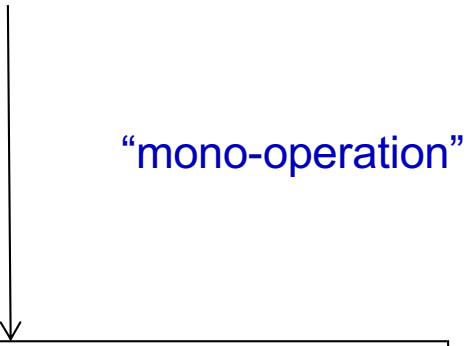
Process and Project (Task) Outcome



How to measure product quality?

Conceptual level

Product quality



Operational (measurable) level

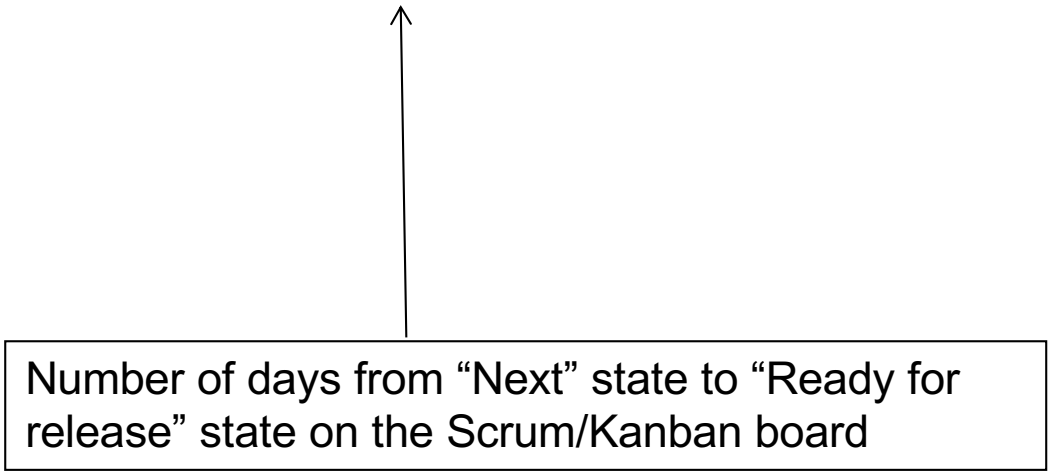
Number of weighted bugs in the severity levels: Blocking (weight 8), Critical (4), Moderate (2), and Minimal (1)

How to measure (lead) time?

Conceptual level

Lead time

Operational (measurable) level



Definition of Lead time

- **Normal definition:**
  - the time from a customer issues a request for a new or changed feature until it is implemented and deployed in the customer's environment
- **In the context of SI**, which is an in-house development company:
  - the time from the team receives the request (state "Next") until it's ready for release (state "Ready for release)

Even for objective data, imprecise definitions may cause different people to measure the phenomenon differently and thus obtain different results

How to measure cost?

Conceptual level

Cost

Productivity

Operational (measurable) level

Alt.1: Number of work items developed per developer per quarter

Alt 2.:Total churn (number of lines added, deleted, or modified) per developer per quarter

Data collection

Information on 12 000 work items over 3.5 years recorded in Team Foundation Server (TFS), now called Azure DevOps Server

	ID	Type	State From	State To	Direction	Created Date	Date From	Date To	Testing Impact	Lead Time	Lines Added	Lines Modified	Lines Deleted	Churn	Process Type
1	72805	Bug	Acceptance-	Ready for Release-	1		2012-11-22	2012-11-22	3 - High	•	•	•	•	•	Kanban
2	72805	Bug	Test-Done	Acceptance-	1		2012-11-22	2012-11-22	3 - High	•	•	•	•	•	Kanban
3	72805	Bug	Test-In Progress	Test-Done	1		2012-11-22	2012-11-22	3 - High	•	•	•	•	•	Kanban
4	72805	Bug	Development-Done	Test-In Progress	1		2012-11-20	2012-11-20	3 - High	•	•	•	•	•	Kanban
5	72805	Bug	Analysis-In Progress	Development-Done	1		2012-11-17	2012-11-19	3 - High	•	•	•	•	•	Kanban
6	72805	Bug	Next-HF/Update	Analysis-In Progress	1	2012-10-26	2012-11-08	2012-11-08	3 - High	15	0	0	0	0	Kanban
7	72806	Bug	Ready for Release-	Released-HF/Update	1		2012-10-29	2012-11-12	3 - High	•	•	•	•	•	Kanban
8	72806	Bug	Acceptance-	Ready for Release-	1		2012-10-29	2012-10-29	3 - High	•	•	•	•	•	Kanban
9	72806	Bug	Test-Done	Acceptance-	1		2012-10-29	2012-10-29	3 - High	•	•	•	•	•	Kanban
10	72806	Bug	Test-In Progress	Test-Done	1		2012-10-29	2012-10-29	3 - High	•	•	•	•	•	Kanban
11	72806	Bug	Development-Done	Test-In Progress	1		2012-10-29	2012-10-29	3 - High	•	•	•	•	•	Kanban
12	72806	Bug	Development-In Progress	Development-Done	1		2012-10-27	2012-10-29	3 - High	•	•	•	•	•	Kanban
13	72806	Bug	Analysis-In Progress	Development-In Progress	1		2012-10-26	2012-10-26	3 - High	•	•	•	•	•	Kanban
14	72806	Bug	Next-HF/Update	Analysis-In Progress	1	2012-10-26	2012-10-26	2012-10-26	3 - High	4	0	0	0	0	Kanban
15	72809	Bug	Ready for Release-HF/Up	Released-HF/Update	1		2012-12-10	2012-12-13	2 - Critical	•	•	•	•	•	Kanban
16	72809	Bug	Test-Done	Ready for Release-HF/Upda	1		2012-12-10	2012-12-10	2 - Critical	•	•	•	•	•	Kanban
17	72809	Bug	Development-Done	Test-Done	1		2012-12-08	2012-12-10	2 - Critical	•	•	•	•	•	Kanban
18	72809	Bug	Development-In Progress	Development-Done	1		2012-12-05	2012-12-05	2 - Critical	•	•	•	•	•	Kanban
19	72809	Bug	Next-In Progress	Development-In Progress	1	2012-10-26	2012-11-29	2012-11-29	2 - Critical	12	2	6	54	62	Kanban
20	72811	Bug	Acceptance-	Ready for Release-	1		2012-11-30	2012-11-30	2 - Critical	•	•	•	•	•	Kanban
21	72811	Bug	Test-Done	Acceptance-	1		2012-11-30	2012-11-30	2 - Critical	•	•	•	•	•	Kanban
22	72811	Bug	Test-In Progress	Test-Done	1		2012-11-29	2012-11-29	2 - Critical	•	•	•	•	•	Kanban
23	72811	Bug	Development-Done	Test-In Progress	1		2012-11-28	2012-11-28	2 - Critical	•	•	•	•	•	Kanban
24	72811	Bug	Development-In Progress	Development-Done	1		2012-11-27	2012-11-27	2 - Critical	•	•	•	•	•	Kanban
25	72811	Bug	Development-Done	Development-In Progress	-1		2012-11-24	2012-11-25	2 - Critical	•	•	•	•	•	Kanban
26	72811	Bug	Analysis-In Progress	Development-Done	1		2012-11-23	2012-11-23	2 - Critical	•	•	•	•	•	Kanban
27	72811	Bug	Next-In Progress	Analysis-In Progress	1	2012-10-26	2012-11-23	2012-11-23	2 - Critical	8	29	172	30	231	Kanban
28	72813	Bug	Ready for Release-HF/Up	Released-HF/Update	1		2012-12-13	2012-12-13	2 - Critical	•	•	•	•	•	Kanban
29	72813	Bug	Test-Done	Ready for Release-HF/Upda	1		2012-12-13	2012-12-13	2 - Critical	•	•	•	•	•	Kanban
30	72813	Bug	Test-In Progress	Test-Done	1		2012-12-11	2012-12-11	2 - Critical	•	•	•	•	•	Kanban
31	72813	Bug	Development-Done	Test-In Progress	1		2012-12-08	2012-12-10	2 - Critical	•	•	•	•	•	Kanban
32	72813	Bug	Analysis-In Progress	Development-Done	1		2012-11-28	2012-11-28	2 - Critical	•	•	•	•	•	Kanban
33	72813	Bug	Next-In Progress	Analysis-In Progress	1	2012-10-26	2012-11-27	2012-11-27	2 - Critical	17	75	63	84	222	Kanban
34	72821	Feature	Test-Done	Ready for Release-	1		2012-11-03	2012-11-05		•	•	•	•	•	Kanban
35	72821	Feature	Test-In Progress	Test-Done	1		2012-11-03	2012-11-03		•	•	•	•	•	Kanban
36	72821	Feature	Next-	Test-In Progress	1	2012-10-26	2012-10-31	2012-10-31		6	0	0	0	0	Kanban
37	72822	Feature	Test-Done	Ready for Release-	1		2012-11-06	2012-11-06		•	•	•	•	•	Kanban
38	72822	Feature	Test-In Progress	Test-Done	1		2012-11-06	2012-11-06		•	•	•	•	•	Kanban
39	72822	Feature	Next-	Test-In Progress	1	2012-10-26	2012-10-31	2012-10-31		7	0	0	0	0	Kanban
40	72823	Feature	Test-Done	Ready for Release-	1		2012-11-03	2012-11-05		•	•	•	•	•	Kanban
41	72823	Feature	Test-In Progress	Test-Done	1		2012-11-02	2012-11-02		•	•	•	•	•	Kanban
42	72823	Feature	Next-	Test-In Progress	1	2012-10-26	2012-11-02	2012-11-02		4	0	0	0	0	Kanban
43	72824	Feature	Test-Done	Ready for Release-	1		2012-11-03	2012-11-05		•	•	•	•	•	Kanban
44	72824	Feature	Test-In Progress	Test-Done	1		2012-10-31	2012-10-31		•	•	•	•	•	Kanban
45	72824	Feature	Next-	Test-In Progress	1	2012-10-26	2012-10-30	2012-10-30		6	0	0	0	0	Kanban
46	72826	Feature	Test-Done	Ready for Release-	1		2012-11-06	2012-11-06		•	•	•	•	•	Kanban
47	72826	Feature	Test-In Progress	Test-Done	1		2012-11-06	2012-11-06		•	•	•	•	•	Kanban
48	72826	Feature	Next-	Test-In Progress	1	2012-10-26	2012-11-03	2012-11-05		4	0	0	0	0	Kanban
49	72827	Feature	Test-Done	Ready for Release-	1		2012-11-04	2012-11-05		•	•	•	•	•	Kanban
50	72827	Feature	Test-In Progress	Test-Done	1		2012-11-03	2012-11-04		•	•	•	•	•	Kanban
51	72827	Feature	Next-	Test-In Progress	1	2012-10-26	2012-11-02	2012-11-02		4	0	0	0	0	Kanban
52	72828	Feature	Test-Done	Ready for Release-	1		2012-11-09	2012-11-09		•	•	•	•	•	Kanban
53	72828	Feature	Test-In Progress	Test-Done	1		2012-11-09	2012-11-09		•	•	•	•	•	Kanban
54	72828	Feature	Next-	Test-In Progress	1	2012-10-26	2012-11-03	2012-11-05		7	0	0	0	0	Kanban
55	72829	Feature	Test-Done	Ready for Release-	1		2012-11-09	2012-11-09		•	•	•	•	•	Kanban
56	72829	Feature	Test-In Progress	Test-Done	1		2012-11-09	2012-11-09		•	•	•	•	•	Kanban
57	72829	Feature	Next-	Test-In Progress	1	2012-10-26	2012-11-08	2012-11-08		2	0	0	0	0	Kanban

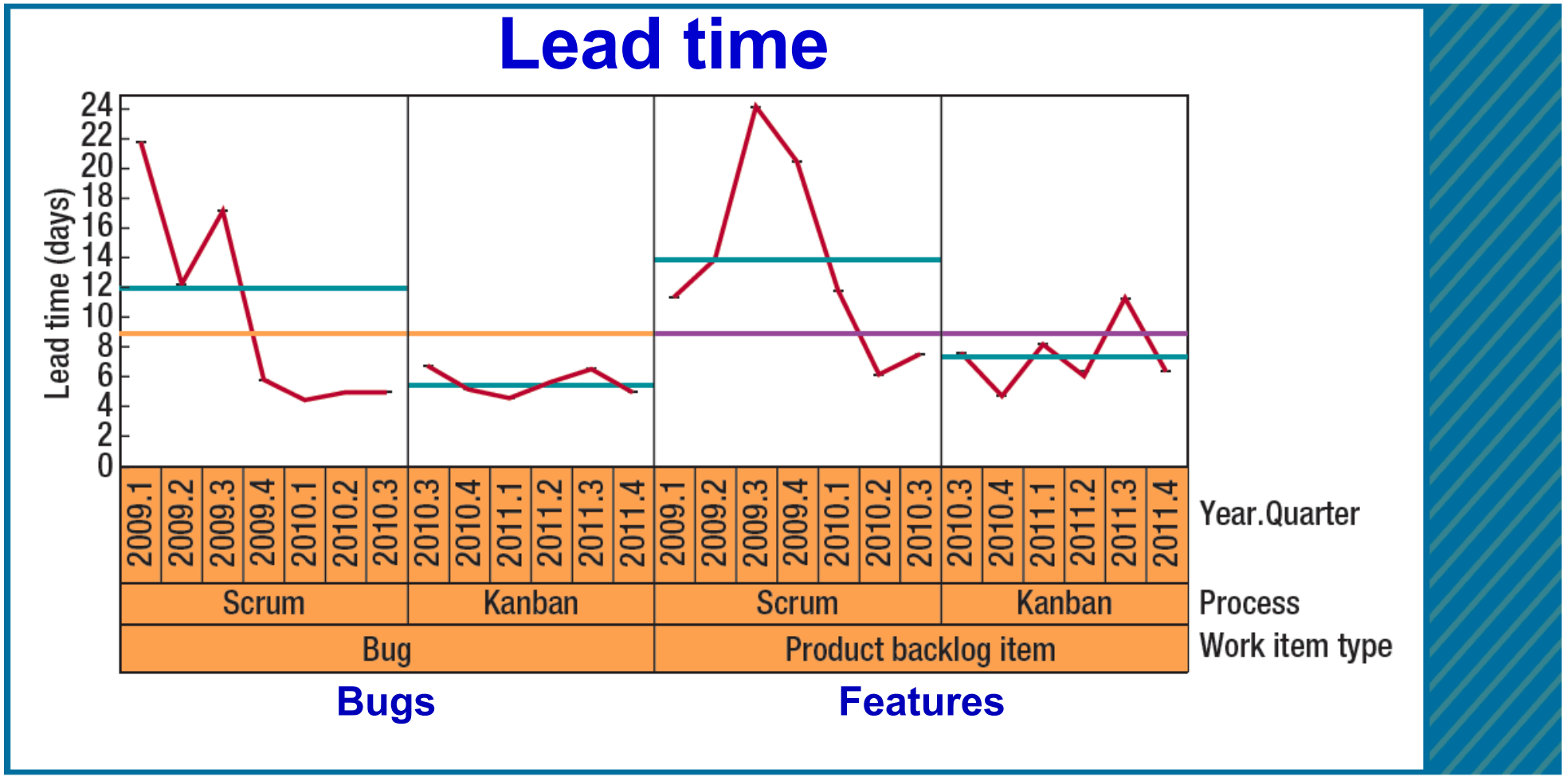


FIGURE 1. Average lead time measured in days by work item type, process, and quarter. The average lead time declined by approximately 50 percent from the Scrum period to the Kanban period. For bugs, the average lead time fell from 12 days for Scrum to five for Kanban. For the project backlog items (PBIs), the lead time declined from 14 to seven days. The orange and purple lines indicate that the bugs and PBIs had the same (weighted) average lead time (nine days) over the whole period. The local top on each third quarter is due to less activity during the summer holiday.

Bugs

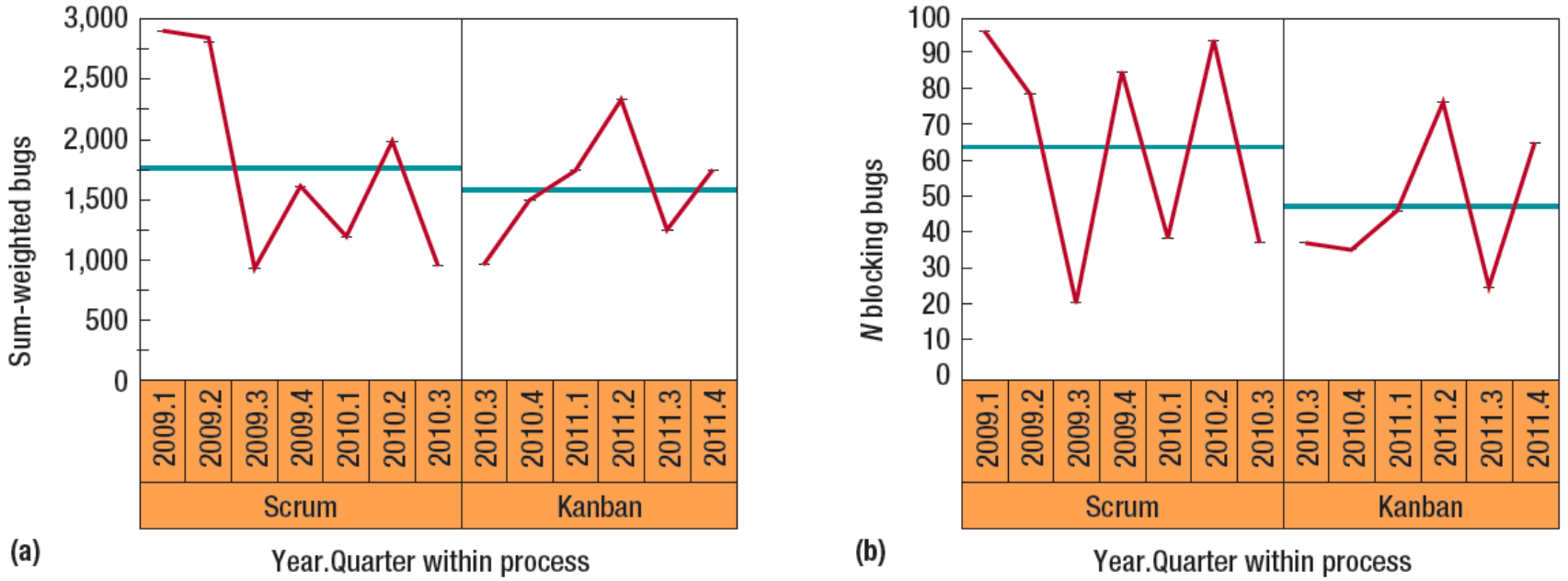


FIGURE 3. Bugs: (a) weighted and (b) blocking. The average number of weighted bugs per quarter fell from 1,774 in the Scrum period to 1,591 in the Kanban period. The most critical bugs, blocking bugs, declined in number even more between the two process periods (from 65 to 48).

Productivity alt. 1

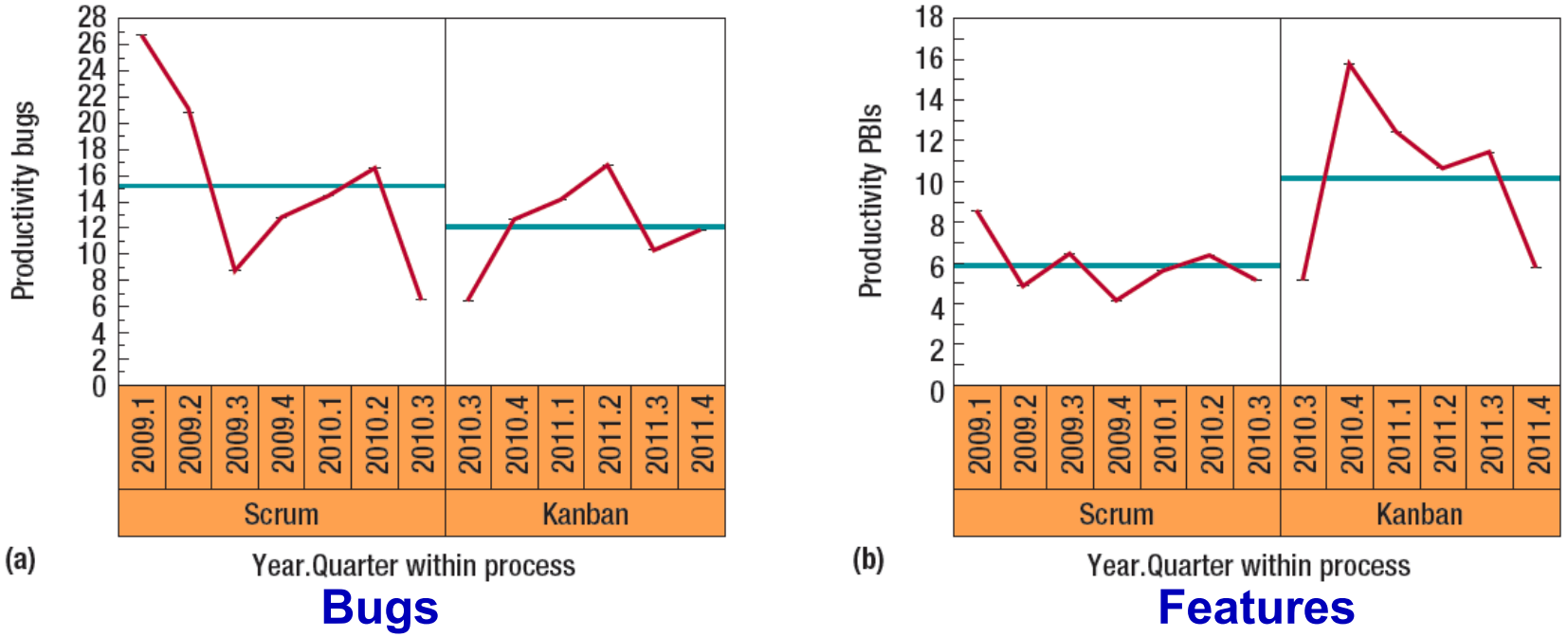


FIGURE 4. Productivity: (a) bugs per developer and (b) PBIs per developer. The number of work items per person, or productivity, decreased from 15.3 to 12.1 for bugs but increased from 5.9 to 10.2 for PBIs.

Moderator variable: Churn

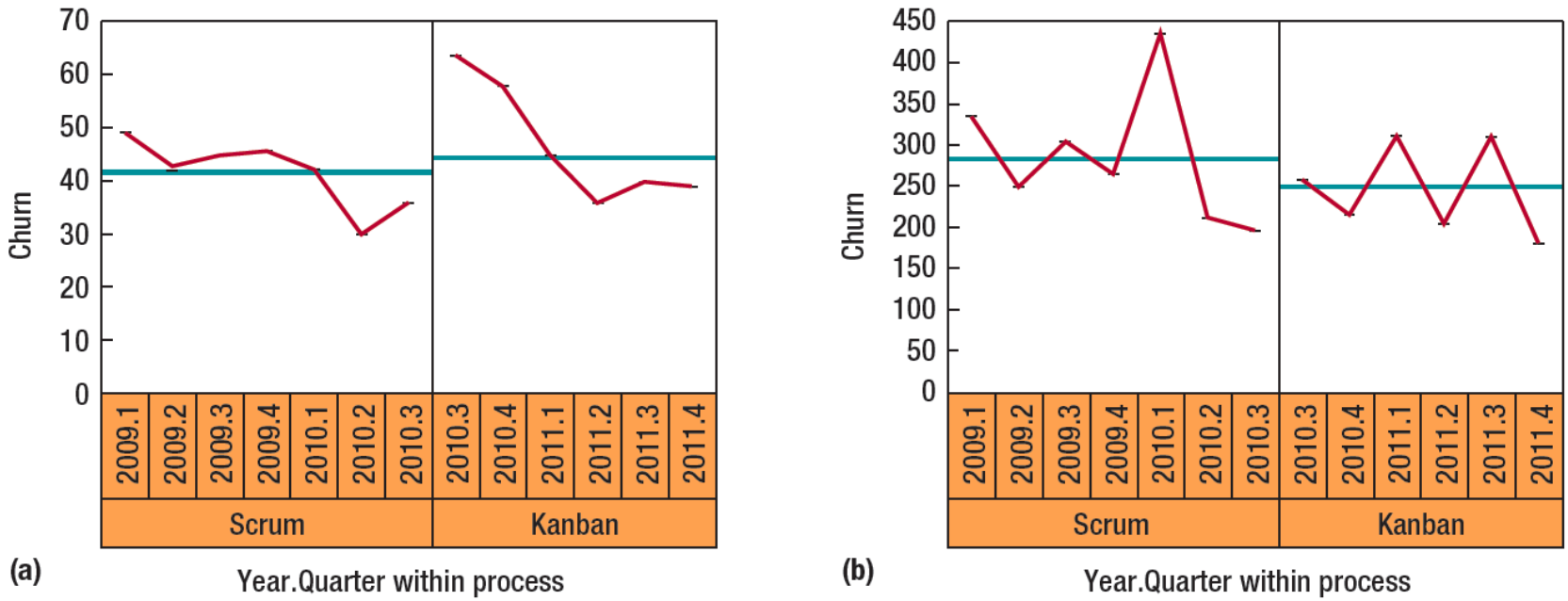


FIGURE 2. Average churn of (a) bugs and (b) PBIs. The average churn for bugs is 6 percent higher in the Kanban period than in the Scrum period, while for PBIs, the average churn is 12 percent lower in the Kanban period than in the Scrum period.

Productivity alt. 2

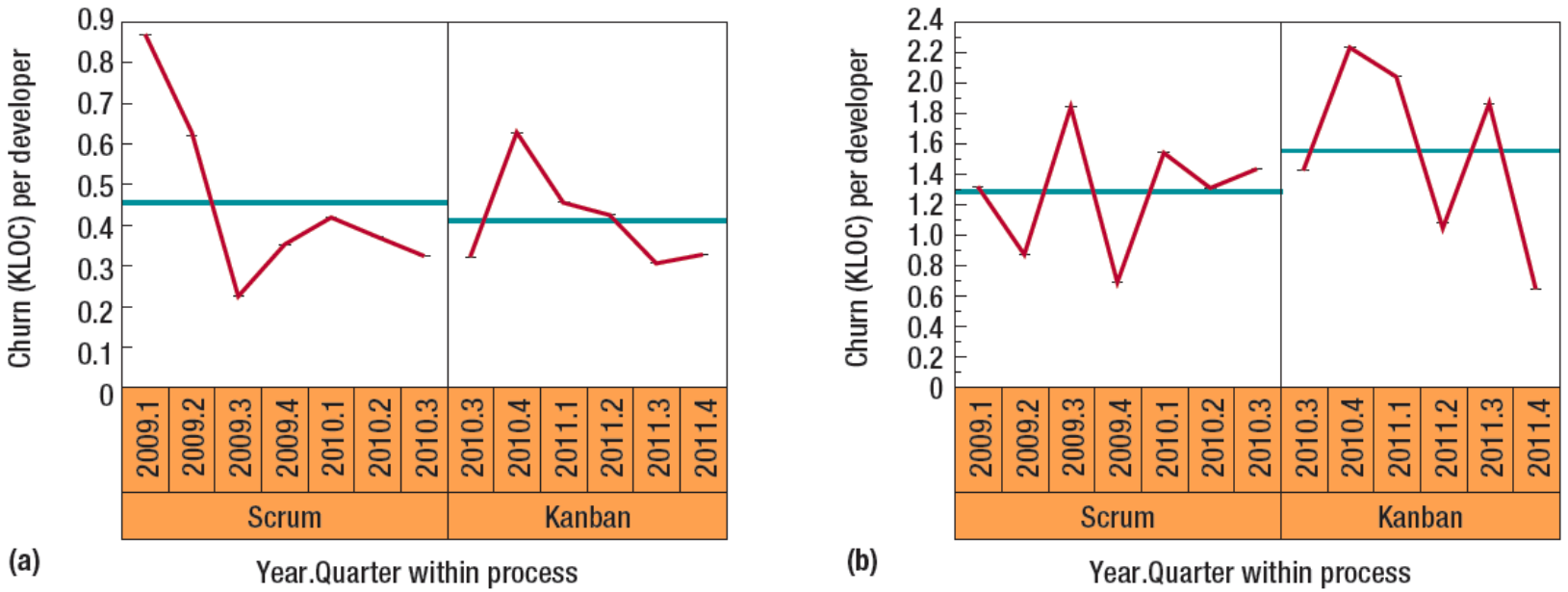


FIGURE 5. Productivity 2: (a) bugs per developer and (b) PBIs per developer. For bugs, productivity decreased from an average of 0.46 KLOC for Scrum to 0.41 KLOC for Kanban; for PBIs, productivity increased from an average of 1.28 KLOC for Scrum to 1.55 KLOC for Kanban.

Qualitative evaluation

- Interviewed: R&D Operations Manager, CTO, one team leader, and one developer
- The fixed timeboxes in Scrum perceived artificial
- Work items frequently underestimated
- Developers have to deal with ad hoc bug fixing, support, and maintenance tasks while working on the items. Still, supposed to finish the items within the given timebox
- The timeline led to work items that were finished before the quality was satisfactory, that were deferred to the next iteration (which required new planning activities), or that were not finished at all. In the Kanban period, the items that had been started were finished because the developers focused on one item at a time until it was finished
- Difficult to allocate the resources optimally within the sprints. For example, the testers tended to have little to do in the beginning of a sprint and too much at the end.
- Much of the sprint start-up meetings were **perceived** as “waste”
- Did the lack of timeboxes in Kanban lead to insufficient pressure to finish items? The **consensus** stated that the combination of daily stand-up meetings and weekly status meetings, the visibility of the items’ status on the board, and the personal ambitions to complete the job constituted sufficient pressure

Summary of variables in the study

	Name	Values
Independent variables	Process	Scrum or Kanban
	Type of work item	Bug or Project Backlog Item (new features, adaptive maintenance tasks, and support tasks, i.e., all tasks that are not bug fixing)
Control variable	Year.Quarter	Each quarter from 2009.1 to 2011.4
	Churn	Number of lines added, deleted, or modified
Dependent variables	Lead time	Number of days from "Next" state to "Ready for release" state on the board
	Production	Number of work items developed per quarter (often called "throughput" [3])
	Productivity	Production per developer
	Productivity 2	Total churn per developer per quarter
	Quality	Number of weighted bugs in the severity levels: Blocking (weight 8), Critical (4), Moderate (2), and Minimal (1)

Summary of study

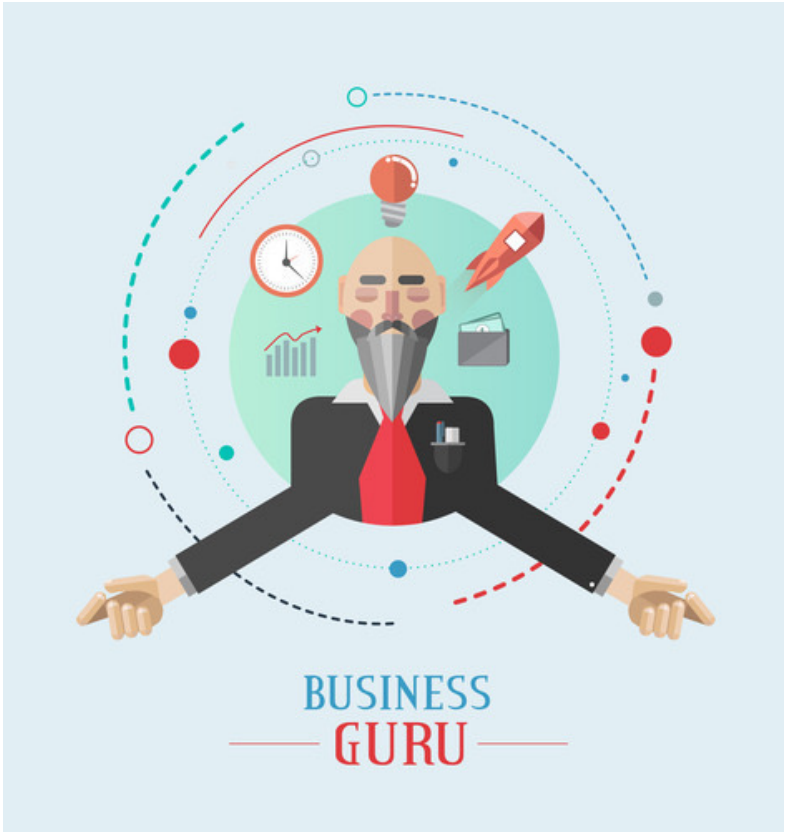
- By replacing Scrum with Kanban, SI
  - Almost halved the lead time
  - Reduced the number of bugs by 10%
  - Improved productivity
- SI appears to benefit from using Kanban over Scrum
- Kanban should be considered by other companies that
  - Difficulties with estimation
  - Interruptions due to ad hoc-bug fixing, support and maintenance tasks

Full report: Dag I.K. Sjøberg, Anders Johnsen and Jørgen Solberg: Quantifying the Effect of Using Kanban versus Scrum: A Case Study. *IEEE Software*, Vol. 29, Nr. 5, pages 47–53, Sep./Oct. 2012

Rethorical traps in agile (book Section 2.2)

- Proof by anecdote
- Slander by association
  - Use negative associations to critize others (“waterfall”). “Agile” is positively loaded; “smidig” in Norwegian, even more
- Intimidation
  - Don’t be worried about being critical about the hype around agile, including asking for evidence for various claims, even though you may be accused of being old-fashioned or even stupid
- Catastrophism
  - “Software development was a catastrophy before agile”; see amusing p. 30 in book
- All-or-nothing (later slide)
- Cover-your-behind
  - ”You should follow this agile principle, but not always” – without specifying when not to follow it
- Unverifiable claims (next slide)

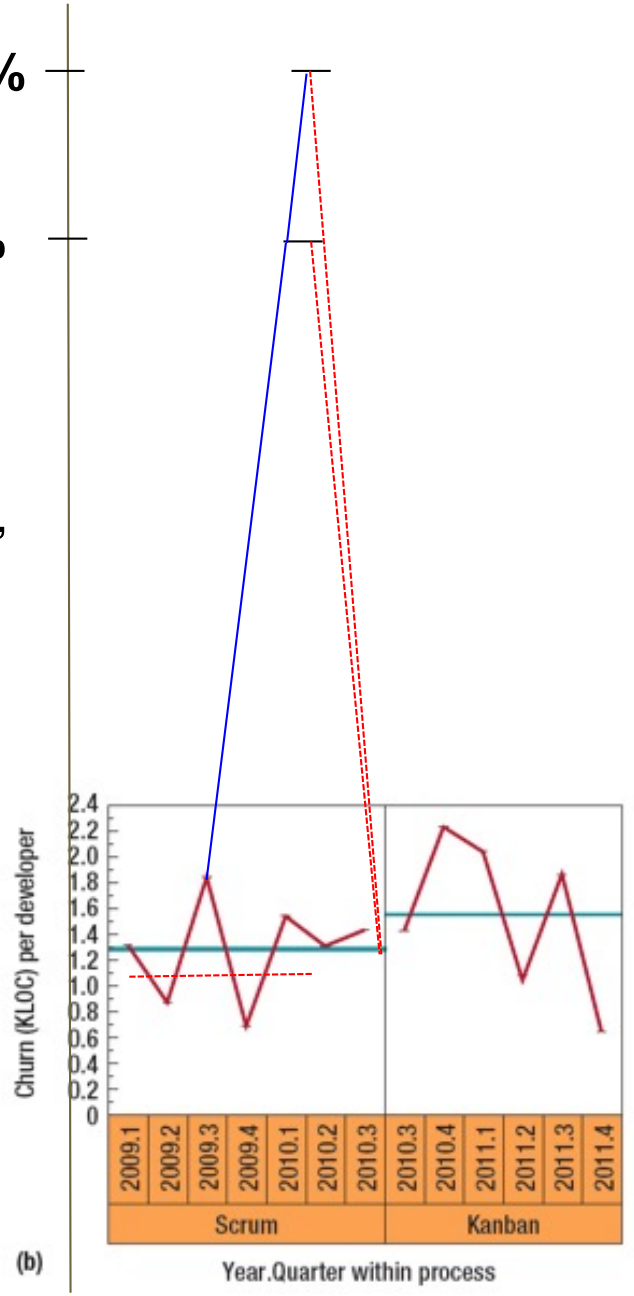
“Unverifiable claims”: Reality versus guru



Guru promise:
After internal Scrum course summer 2009: ”500-800 % more productive in about a year”

ref. Schwaber of 90 % productivity gains, book p. 28

Guess: who was the guru?
See book p. 30, “you have been ill served by the software industry”



“All-or-nothing”
Guru: You didn’t do it exactly the way I said you should

Fra: ….
Sendt: lø 04.02.2012 15:31
Til: Dag Sjøberg
Kopi: ...
Emne: Re: I am allowed to show this to individuals but to emphasize it is under

Dag,

Over 50% of teams saying they are using Scrum do not have a shippable increment of code (fully tested) at the end of the sprint. This will decrease velocity by at least a factor of 2 (we have seen it as high as 24 times longer to finally product shippable code). I suspect Scrum improvement may be related to this.

Furthermore, good Scrum teams (1) make all work visible, (2) limit work in progress, and (3) measure cycle time (which is Kanban). None of this is discussed in the paper.

Regards,

Response from Software Innovation:

Hi

The paper had a word limit. We would probably explain how we implemented scrum and how we currently implement KanBan if we had more "space" available. Generally we can say that when we moved to KanBan, we took what we felt was the best with Scrum (and removed what we felt was not so good), and coupled it with KanBan. The differences in how we implemented the two can be documented and explained in detail, of course.

-Regarding making work visible; we have steadily increased and improved visibility from 2007 until now. From about 2009 we have had automated reports and taskboards for all teams. So we did this well for both Scrum and KanBan, but probably better for KanBan.

-Regarding work in progress; I agree that good scrum teams handle WIP (because it is really up to the team...), but one of the huge advantages of the KanBan board is that the mediocre teams are forced handle wip as well.

-Shippable code; the PBIs and bug fixes implemented during KanBan have been just as shippable as the PBIs were during Scrum. There is no difference in the quality gates for an item moving from the backlog to released between Scrum and KanBan.

Regards,

“Shippable increment of code (fully tested)”

- Each project must provide a definition of when a code increment is done, book Section 8.6. Examples book (increasing level of difficulty):
  - releasable (= “shippable” on prev. slide),
  - unit- and integration-tested; ready for acceptance test; deployed on a demo server, or
  - acceptance-tested; release notes written; releasable; no increased technical debt (lecture 25 October)
- Some discussion of “shippable” (almost done) vs. ”releasable” (fully done, put into production)

Next week:
Lean and agile software engineering