SLIDE PRINTOUT

This document includes slides from Bertrand Meyer's STAF 2014 keynote

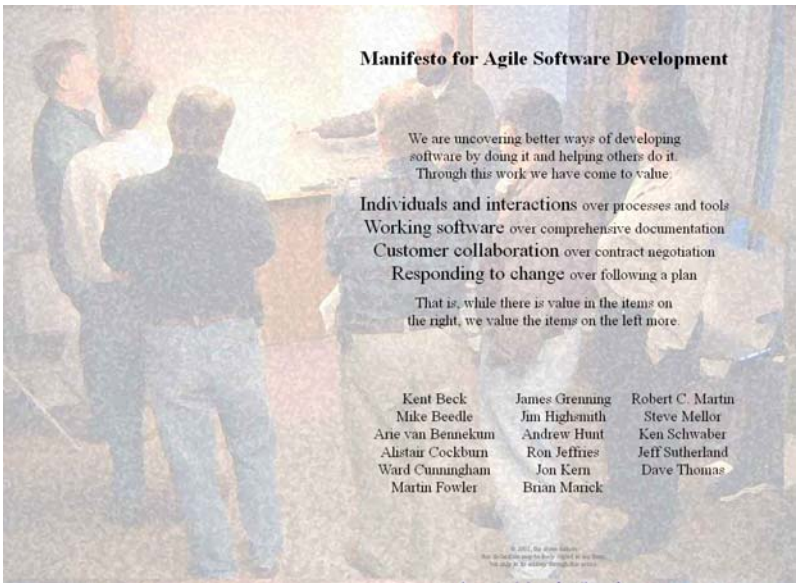
"*Agile! The Good, the Hype and the Ugly*"

(after eponymous book, Springer, 2014; see registration desk for flyer with special discount for STAF attendees)

The document is meant solely for the personal use of STAF 2014 participants

Please note that some of the material (e.g. Agile Manifesto) may be others' copyright. Slides with known distribution restrictions have been removed

Agile manifesto



Manifesto for Agile Software Development

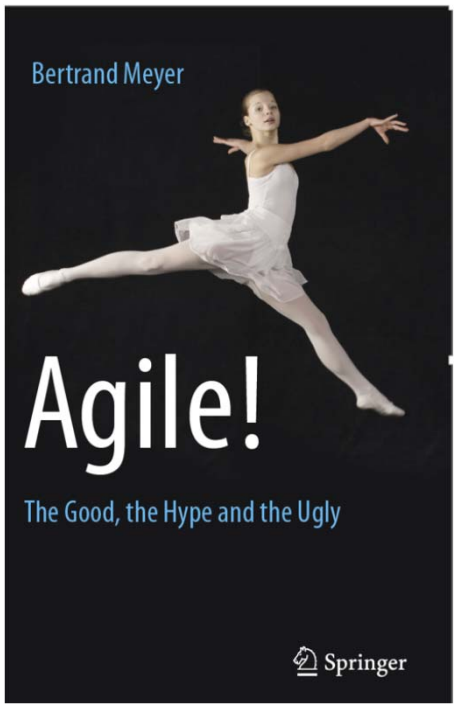
We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

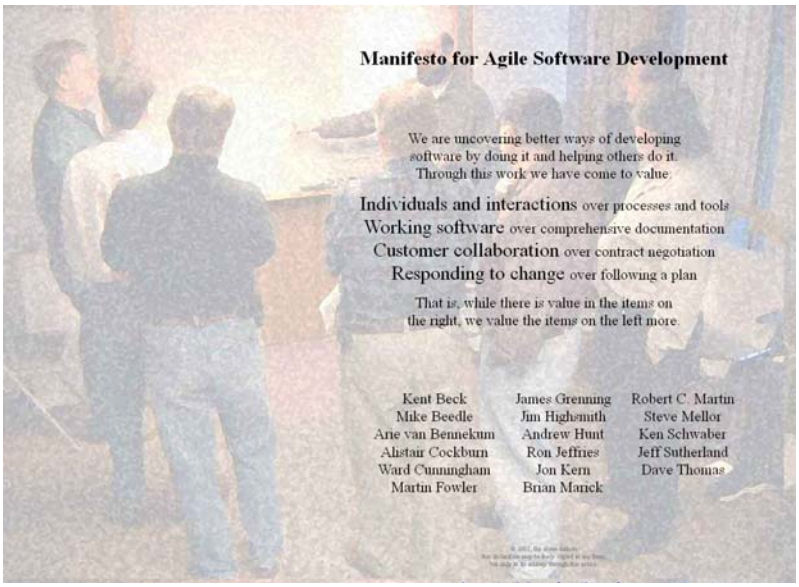
That is, while there is value in the items on the right, we value the items on the left more.

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

-



Agile manifesto



General observations

Your work, Sir, is both new and good, but what's new is not good and what's good is not new

Samuel Johnson*

For every complex problem there is an answer that is clear, simple, and wrong

H.L. Mencken

ANDROMAQUE: I do not understand abstractions.
CASSANDRA: As you like. Let us resort to metaphors.

Jean Giraudoux, *The Trojan War Will Not Happen*

* (probably apocryphal)

Agile methods

Kent Beck

Mary Poppendieck

Alistair Cockburn

Schwaber & Sutherland

Sociological view

"The revolt of the cubicles"

[Dilbert's and Dilbert's boss's pictures removed]

Negotiated scope contract

XP

Source: Beck 2005

"Write contracts for software development that fix time, costs, and quality but call for an ongoing negotiation of the precise scope of the system. Reduce risk by signing a sequence of short contracts instead of one long one."

"You can move in the direction of negotiated scope. Big, long contracts can be split in half or thirds, with the optional part to be exercised only if both parties agree. Contracts with high costs for change requests can be written with less scope fixed up front and lower costs for changes"

Rhetorical devices

- **Proof by anecdote**
- **Slander by association**
- **Intimidation**
- **All-or-nothing**
- **Unverifiable claims**

Proof by anecdote

Source: Mike Cohn, *Succeeding with Agile**

There is a grand myth about requirements — if you write them down, users will get exactly what they want. That's not true. At best, users will get exactly what was written down, which may or may not be what they want. Written words are misleading — they look more precise than they are.

For example, recently I wanted to run a public training course. I sent my assistant an e-mail "*Please book the Denver Hyatt*". The next day she e-mailed, "*the hotel is booked*". I e-mailed back "*Thanks*".

A week later she e-mailed "*the hotel is booked on the days you wanted. What do you want to me do? Do you want to try another hotel in Denver? A different week? A different city?*". We had miscommunicated about the meaning of "booked". When she wrote "*the hotel is booked*", she meant "The room we usually use at the Hyatt is already taken". When I read "*the hotel is booked*", I took it as a confirmation that she had booked the hotel. Neither of us did anything wrong. Rather, this is an example of how easy it is to miscommunicate, especially with written language. If we had been talking rather than e-mailing, I would have thanked her when she told me "*the hotel is booked*". The happy tone of my voice would have confused her, and we would have caught our miscommunication right then.

*Slightly abridged

Intimidation

Source: Steve Denning, *Forbes*, April 2012

[Objection] *"Agile was designed for experienced, smart, and high-achieving people, who would succeed with any project. Not every group can be thus motivated, experienced, and skilled. We have to work with the staff we have. So Agile is not for us."*

In other words, make do with mediocrity. Learn to live with the people who are either not experienced or smart or high-achieving. Hierarchical bureaucracy makes an assumption of incompetence and expects mediocre performance. It learns to live with mediocrity on a permanent basis, in the process creating—not surprisingly—the need for the layers middle managers to provide the "close supervision".

Agile makes the opposite assumption: competence. […] Agile is a way of forcing either high performance or change.

Agile squeezes out mediocrity and requires high-performance. Hierarchical bureaucracy breeds incompetence and feeds off mediocrity: the organization performs accordingly. Faced with the choice between high-performance and the mediocrity, traditional management opts for mediocrity.

More of Denning

" When the culture doesn't fit Agile, the solution is not to reject Agile. The solution is to change the organizational culture. One doesn't even have to look at the business results of firms using hierarchical bureaucracy to know that they are fatally ill."

- (Brecht: *" The people have forfeited the confidence of the government; would it not be easier for the government to dissolve the people, and elect another?"*)

Slander by association

Source: Schwaber & Sutherland, Scrum book

- *Although the predictive, or waterfall, process is in trouble, many people and organizations continue to try to make it work.*

and later in the same paragraph:

- *[A customer was using] services from PricewaterhouseCoopers (PWC). The PWC approach was predictive, or waterfall.*

The book's index entry for "*Predictive process*" reads "*See Waterfall*"

Catastrophism

Source: Schwaber & Sutherland, Scrum book

"You have been ill served by the software industry for 40 years—not purposely, but inextricably. We want to restore the partnership."

Also: many agile authors cite the Standish report

All-or-nothing

Two schools in e.g. Scrum

Extreme Programming 1

Source: Beck 2000

To some folks, XP seems like just good common sense. So why the "extreme" in the name? XP takes commonsense principles and practices to extreme levels.

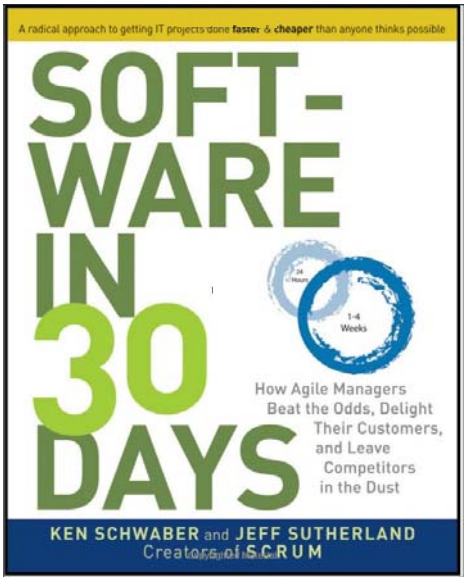
- If code reviews are good, we'll review code all the time (pair programming).
- If testing is good, everybody will test all the time (unit testing), even the customers (functional testing).
- If design is good, we'll make it part of everybody's daily business (refactoring).
- [etc.]

Extreme Programming 2

Source: Beck 2005

There are better ways and worse ways to develop software. Good teams are more alike than they are different. No matter how good or bad your team you can always improve.

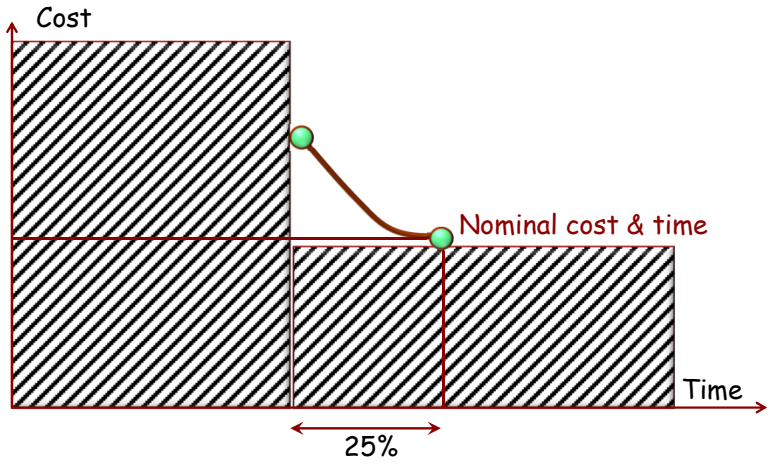
Unverifiable claims



Claims

Reminder: software engineering has laws

Example: Boehm, McConnell, Putnam, Capers Jones…



Ways to assess methodology matters

Intuitive (gut feeling)

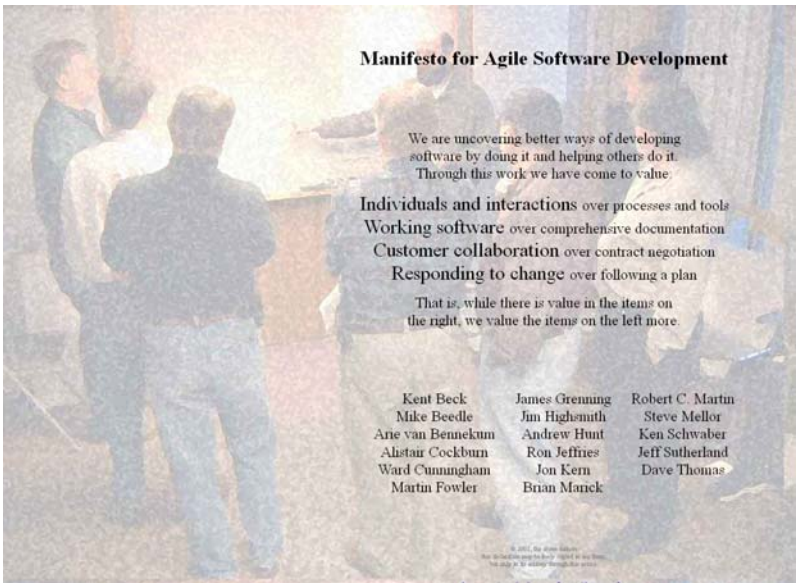
Recently I discovered why the use of the go to statement has such disastrous effects, and I became convinced that it should be abolished from all higher-level programming languages

Experiential (including anecdotal)

Logical (analytic)

Empirical

Agile manifesto



Twelve principles

Source: Agile manifesto

We follow these principles:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Wrong

Redundant

Redundant

What about testing?

Practice

Assertion

Assertion

Assertion

Wrong

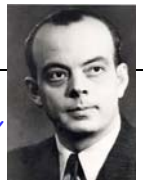
Practice

Finishing a design

It seems that the sole purpose of the work of engineers, designers, and calculators is to polish and smooth out, lighten this seam, balance that wing until it is no longer noticed, until it is no longer a wing attached to a fuselage, but a form fully unfolded, finally freed from the ore, a sort of mysteriously joined whole, and of the same quality as that of a poem. It seems that perfection is reached, not when there is nothing more to add, but when there is no longer anything to remove.

(Antoine de Saint-Exupéry,
Terre des Hommes, 1937)

Finishing a design



Il semble que tout l'effort industriel de l'homme, tous ses calculs, toutes ses nuits de veille sur les épures, n'aboutissent […] qu'à la seule simplicité, comme s'il fallait l'expérience de plusieurs générations pour dégager peu à peu la courbe d'une colonne, d'une carène, ou d'un d'avion, jusqu'à leur rendre la pureté élémentaire de la courbe d'un sein ou d'une épaule. Il semble que le travail des ingénieurs, […] des calculateurs du bureau d'études ne soit ainsi, en apparence, que de polir et d'effacer, d'alléger […] Il semble que la perfection soit atteinte non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher.

(Antoine de Saint-Exupéry,
Terre des Hommes, 1937)

Steve Jobs, 1998

That's been one of my mantras -- focus and simplicity. Simple can be harder than complex: You have to work hard to get your thinking clean to make it simple. But it's worth it in the end because once you get there, you can move mountains.

Twelve principles

Source: Agile manifesto

We follow these principles:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Wrong

Redundant

Redundant

What about testing?

Practice

Assertion

Assertion

Assertion

Wrong

Practice

Agile values

- **A** New, reduced role for manager
- **B** No "Big Upfront" steps
- **C** Iterative development
- **D** Limited, negotiated scope
- **E** Focus on quality, achieved through testing

Agile principles

Organizational

- **1** Place the customer at the center
- **2** Develop minimal software:
  - 2.1 Produce minimal functionality
  - 2.2 Produce only the product requested
  - 2.3 Develop only code and tests
- **3** Accept disciplined change
  - 3.1 Do not change requirements during an iteration
- **4** Let the team self-organize
- **5** Maintain a sustainable pace

Technical

- **6** Produce frequent working iterations
- **7** Treat tests as a key resource:
  - 7.1 Do not start any new development until all tests pass
  - 7.2 Test first
- **8** Express requirements through scenarios

Agile roles

Customer

Team

Product owner

Scrum Master

Agile practices

Managerial:

- Sprint
- Daily meeting
- Planning game, planning poker
- Onsite customer
- Open space
- Collective code ownership

Technical:

- Pair programming
- Daily build
- Continuous integration
- Refactoring
- Test-driven & test-first development

Agile artifacts 1

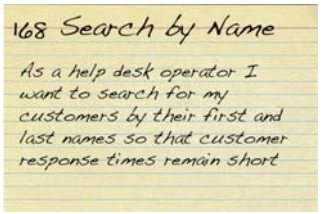
Virtual:

- Code
- Tests
- User stories
- Story points
- Velocity
- Definition of Done
- Working space
- Product backlog, iteration backlog

Agile artifacts 2

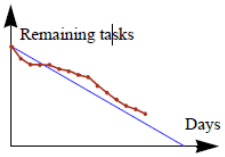
Physical:

- Story card, task card



- Task board

- Burndown chart



- Waste, technical debt

Additive and multiplicative complexity

Adding features

Source: Pamela Zave

Historically, developers of telecommunication software have had no effective means of understanding and managing feature interactions. As a result, feature interactions have been a notorious source of runaway complexity, software bugs, cost and schedule overruns, and unfortunate user experiences. Developers of other software systems are beginning to realize that they, too, have a feature-interaction problem

Consider "busy treatments" in telephony, which are features for handling busy situations by performing functions such as forwarding the call to another party, interrupting the callee, retrying the call later, or offering voice mail to the caller. Suppose that we have a feature-description language in which a busy treatment is specified by providing an action, an enabling condition, and a priority. Further suppose that a special feature-composition operator ensures that, in any busy situation, the single action applied will be that of the highest-priority enabled busy treatment.

In a busy situation where two busy treatments B1 and B2 are both enabled, with B2 having higher priority, these features will interact: the action of B1 will not be applied, even though its stand-alone description of B1 says that it should be applied.

Zave (continued)

Bob has the "call-forwarding" feature enabled and is forwarding all calls to Carol. Carol has "do-not-disturb". Alice calls Bob, the call is forwarded to Carol, and Carol's phone rings, because "do-not-disturb" is not applied to a forwarded call.

Alice calls a sales group. A feature for the sales group selects Bob as a sales representative on duty, and forwards the call to Bob. Bob's cellphone is turned off, so his personal Voice Mail answers the call and offers to take a message. It would be much better to re-activate the sales-group feature to find another representative.

A new Mobility service is offered to office workers. When Alice signs up, her office phone number is forwarded to the Mobility service. On receiving a call for Alice, the Mobility service forwards it to wherever Alice's personal data dictates. However, whenever the data indicates that Alice is in her office, an incoming call enters a forwarding loop.

User stories (imagined)

(#1) **As an** executive, **I want** a redirection option **so that** if my phone is busy the call is redirected to my secretary

…

(#2) As a system configurator, I want to be able to specify various priorities for "busy" actions

..

(#3) As a salesperson, I want to make sure that if a prospect calls while I am in a conversation, the conversation is interrupted **so that** I can take the call immediately

…

(#4) As a considerate correspondent, I want to make sure that if a call comes while my phone is busy I get to the option of calling back as soon as the current call is over

Daily meeting

Scrum

Goal: to set the day's work

Held every morning

Time-limited, usually 15 minutes

Involves all team members, with special role for those who are "committed" (over those just "involved")

Enables every team member to answer three questions:

- What did you do yesterday?
- What will you do today?
- Are there any impediments in your way?

Focus on commitments and on uncovering impediments (responsibility of the Scrum Master)

The resolution will take place outside of the meeting

Pair programming

XP

Source: Beck 2005

Two programmers sitting at one machine

Dialog between two people, with shared keyboard & mouse

Goals:

- Keep each other on task
- Brainstorm refinements to system
- Clarify ideas
- Take initiative when other stuck, lowering frustration
- Hold each other accountable to team practices

Test-Driven Development

Standard cycle:

- Add a test
- Run all tests and see if the new one fails
- Write some code
- Run the automated tests and see them succeed
- Refactor code

Expected benefits:

- Catch bugs early
- Write more tests
- Drive the design of the program
- Replace specifications by tests
- Use debugger less
- More modular code
- Better coverage
- Improve overall productivity

Test-Driven Development

The basic idea is sound…

… but not the replacement of specifications by test

Major benefit: keep an up-to-date collection of regression tests

Requirement that all tests pass can be unrealistic (tests degrade, a non-passing test can be a problem with the test and not with the software)

Basic TDD idea can be applied with specifications! See Contract-Driven Development

Planning poker

Scrum

- Present individual stories for estimation
- Discuss
- Each participant chooses from his deck the numbered card that represents estimate of work involved in story under discussion
- Deck has successive numbers (Fibonacci: 1, 2, 3, 5, 8, 13, 20, …)
- Keep estimates private until each participant has chosen a card
- Reveal estimates
- Repeat until consensus

(Variant of Wideband Delphi technique.)

Task board

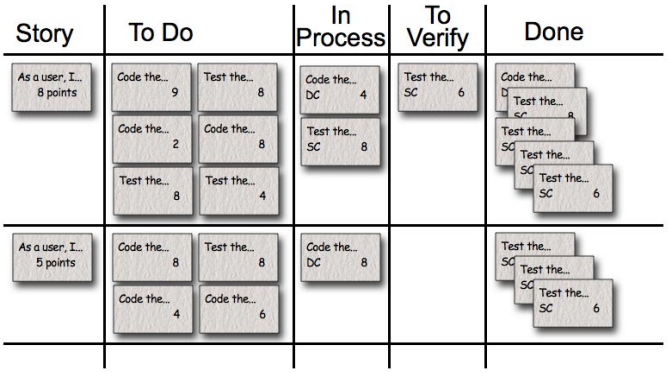
Scrum

Source: Cohn, Anand

Used to see and change the state of the tasks of the current sprint: "to do", "in progress", "done".

Benefits:

- Transparency
- Collaboration
- Prioritization
- Focus
- Self-organization
- Empiricism.
- Humility

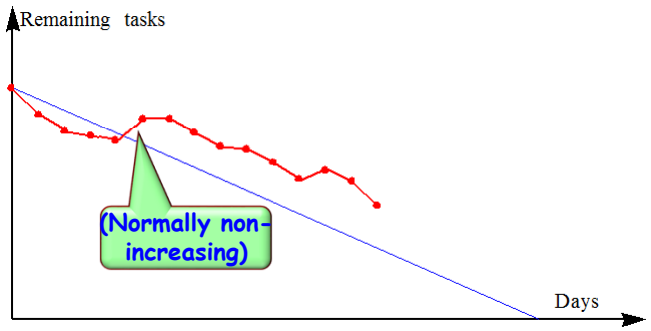


Burndown chart Scrum

Publicly displayed chart, updated every day, showing

- Remaining work
- Progress

in the Sprint backlog



The ugly

- Rejection of upfront tasks
- Particularly: no upfront requirements
- User stories as a replacement for abstract requirements
- Tests as a replacement for specifications
- Feature-based development & ignorance of dependencies
- Dismissal of dependency-tracking tools
- Embedded customer
- Coach & method keeper (e.g. Scrum Master) as a separate role
- Test-driven development
- Dismissal of traditional manager tasks
- Dismissal of auxiliary products
- Dismissal of a priori concern for extendibility
- Dismissal of a priori concern for reusability
- Dismissal of a priori architecture work
- Dismissal of non-shippable artifacts

The indifferent

- Pair programming
- Open-space working arrangements
- Self-organizing teams
- Maintaining a sustainable pace
- Producing minimal functionality
- Planning game, planning poker
- Cross-functional teams

The good

- Acceptance of change
- Frequent iterations
- Emphasis on working code
- Tests as one of the key resources of the project
- Constant test regression analysis
- No branching
- Product (but not user stories!) burndown chart
- Daily meeting

The brilliant

- Short iterations
- Closed-window rule
- Velocity
- Refactoring
- Associating a test with every piece of functionality
- Continuous integration

For more