



IBM Global Services

Estimation in Agile Projects

Dr. Christoph Steindl
Senior IT Architect and Method Exponent
christoph_steindl@at.ibm.com

Pål Krogdahl
Senior IT Architect and Method Exponent
pal.krogdahl@se.ibm.com

Goals of this presentation

- Explain (just) a little bit what “Agile” means.
- Describe estimation and NOT planning in agile projects.
- At the conclusion of this session, you should be able to understand:
 - There are different processes for “Predictable Manufacturing” and “New Software Development”.
 - Estimation in a highly dynamic environment is done differently
 - How estimation is done in an agile way
 - What the reasons are for doing it that way

Motivation

- Plans are only as good as the estimates, that the plans are based on, and estimates always come second to actuals. The real world has this horrible habit of destroying plans.
- The customer has the right to an overall plan, to see progress and to be informed of schedule changes. Whereas the developer has the right to make and update his own estimates and to accept responsibility instead of having responsibility assigned to him.
- You can't put 10 pounds of (whatever) into a 5 pound bag.
- Forecasting tomorrow's weather is much more difficult than telling what the weather was like yesterday.
- Don't try to be too sophisticated; estimates will never be anything other than approximate, however hard you try.

From: Kent Beck and Martin Fowler: Planning Extreme Programming

Software is New Product Development

Most software is not a predictable or mass manufacturing problem. Software development is new product development.

| Predictable Manufacturing | New Product Development |
|--|--|
| It is possible to first complete specifications, and then build. | Rarely possible to create upfront unchanging and detailed specs. |
| Near the start, one can reliably estimate effort and cost. | Near the beginning, it is not possible. As empirical data emerge, it becomes increasingly possible to plan and estimate. |
| It is possible to identify, define, schedule, and order all the detailed activities. | Near the beginning, it is not possible. Adaptive steps driven by build-feedback cycles are required. |
| Adaptation to unpredictable change is not the norm, and change-rates are relatively low. | Creative adaptation to unpredictable change is the norm. Change rates are high. |

A “waterfall” lifecycle, big up-front specifications, estimates, and speculative plans applicable to predictable manufacturing have been misapplied to software projects, a domain of inventive, high-change, high-novelty work.

From: Craig Larman: Agile & Iterative Development

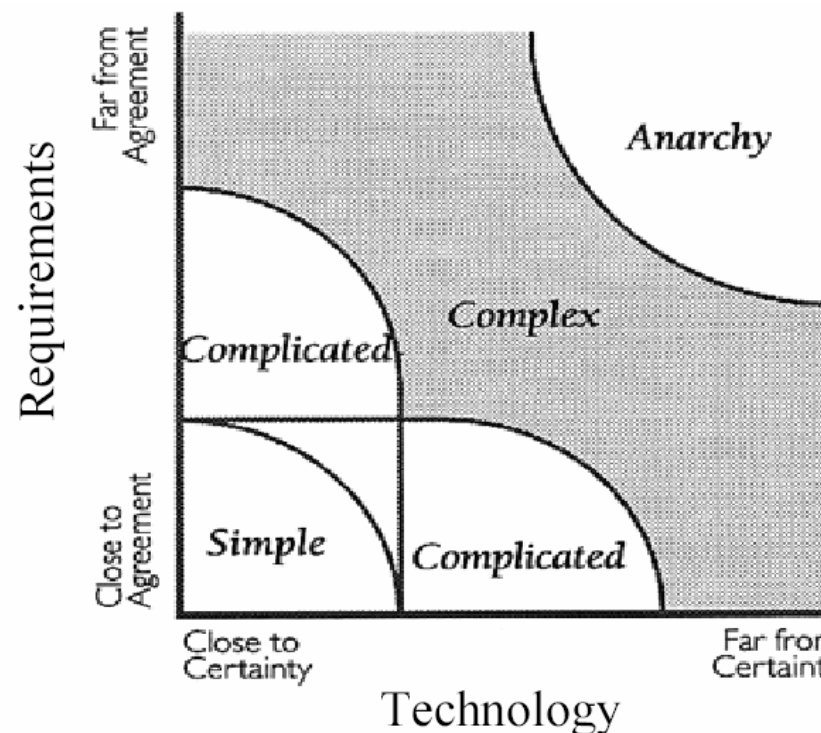
Complexity in development projects

- „It is typical to adopt the **defined** (theoretical) modeling **approach** when the underlying mechanisms by which a process operates are reasonably well understood. When the process is too complicated for the defined approach, the **empirical approach** is the appropriate choice.“

- Empirical Processes:

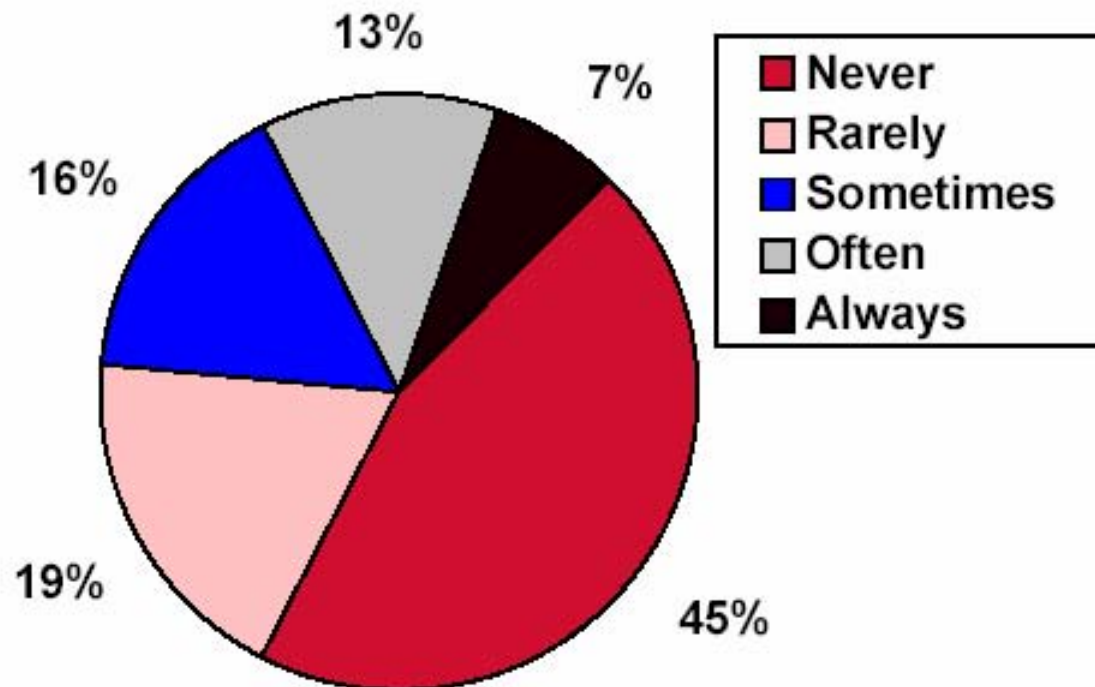
- When you can't define things enough so that they run unattended and produce repeatable, acceptable quality output.
- Empirical models are used when the activities are not predictable, are non-linear, and are too complex to define in repeatable detail.
- Small changes in the input can lead to huge changes in the output („butterfly effect“, chaos theory)
- Control is through inspection and adaptation.

Ogunnaike and Ray: *Process Dynamics, Modeling and Control*.



Features & Function Usage

**THE
STANDISH
GROUP**



<http://www.xp2003.org/xp2002/talksinfo/johnson.pdf>

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, 2001.

Agile Methods are in Wide-Spread Use

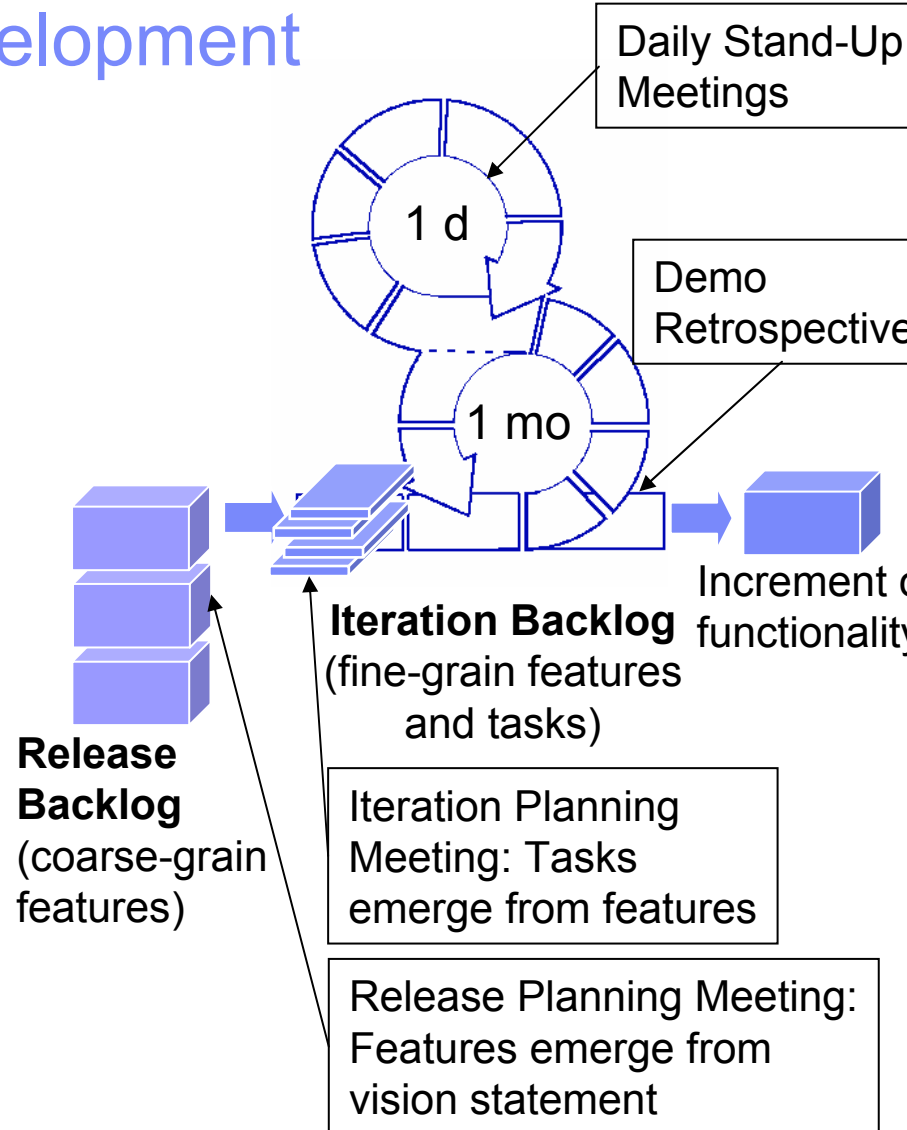
- Extreme Programming (Kent Beck)
- Scrum (Ken Schwaber, Jeff Sutherland, Mike Beedle)
- Crystal (Alistair Cockburn)
- DSDM (Arie van Bennekum)
- Feature-Driven Development (Jeff De Luca)
- Lean Development (Bob Charette)
- Adaptive Software Development (Jim Highsmith)

- There are tons of books on the subject.

- Google reports 49.700 hits for „Agile methods“.

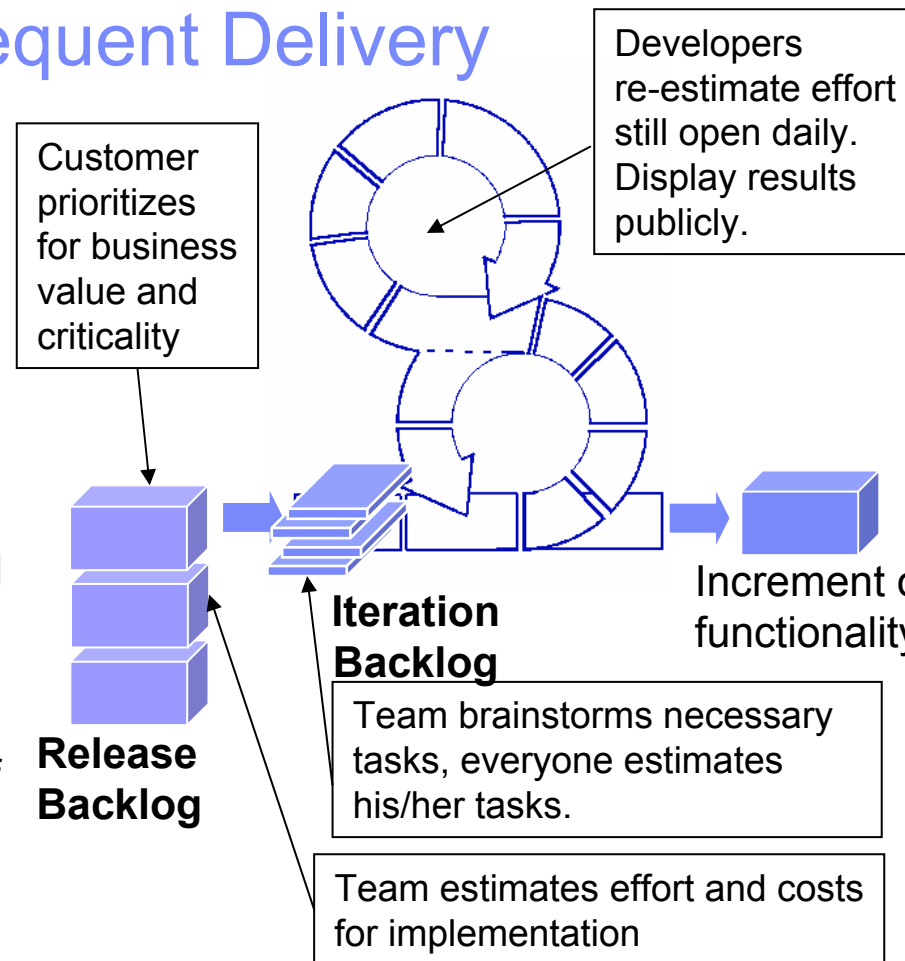
Iterative & Incremental Development

- Frequent inspection and adaptation
 - Daily within team, monthly with customer
- Collaboration, close communication
 - Daily Stand-up Meetings within team, where the customer may attend, but not intrude
- Frequent delivery
 - Demo increment of potentially shippable functionality
- Reflective improvement
 - At the end of each iteration, consider what worked well, what did not work well, what to try next iteration
- (Incremental) Emergence of requirements, technology, and team capabilities
- Empowerment and self-organization
 - Team makes up the tasks to deliver the functionality
 - Team estimates the effort and cost
- Dealing with reality, not artefacts
 - Deliver software (mainly) plus additional requested deliverables



Iterative Estimation with Frequent Delivery

- The team understands - by and by – its velocity, how much it can deliver in an iteration.
- Team members sign up for tasks, there is no project manager who assigns the tasks.
- Everyone estimates his/her tasks, there is no estimation guru who estimates all tasks for all people. The quality of the estimate is commensurate with the information available.
- Everyone estimates often: at the beginning of the iteration, daily during the iteration to estimate the remaining effort.
- The effort remaining (and not the effort already spent) is displayed publicly to enable collaborating teams that work together to meet the target of the iteration.



Agile Principles for Estimation

- If estimating is difficult, the agile approach increases the feedback
 - shorten the time from estimating to feedback about accuracy of estimate
 - increase the frequency of estimating
 - sketch out options and get feedback from the customer before doing detailed estimating
 - The guideline here is: Don't estimate too far into the future, if the future is unclear!
- If the requirements and assumptions are not really clear, the team develops multiple estimates (“Options Thinking”):
 - communicate the constraints / assumptions of the estimates rather than just the numbers
 - discuss the constraints / assumptions with the customer and the customer can give feedback to better align the team's understanding with the business drivers.
- Validate estimates by comparing them with other estimates / experience, use simple rules, triangulation and intuitive decision making.
- The agile approach relies on self-organization of the team, continuous learning and emergence of estimates (besides requirements and design).

Estimating Release Backlog (1/2)

- To gain an understanding of the amount of effort to develop the product or system (**long-term view**).
- Make **more effort** to reach this estimate **for higher priority** than lower priority items, since the lower priority items may change prior to being developed.
- Don't spend too much time on estimating. The goal of the estimates is to **gain a general understanding** of the cost of the system or product. This is used to determine whether it is economic to develop it. Be aware of diminishing returns. Do enough, but not too much.
- The Product Owner works with the **business** departments **and** the **development** organization to develop estimates (to analyze, design, develop, document, and test the functionality, i.e. **whole lifecycle**), usually in person days.
- Use the people who will be staffing the project teams to make the estimates. If they aren't known yet, have people of equal skills and project domain knowledge make the estimates. **Do not have experts or outsiders make the estimates**. Their estimates are irrelevant to those who will actually be doing the work.

From: Ken Schwaber: *Scrum Methodology*

Estimating Release Backlog (2/2)

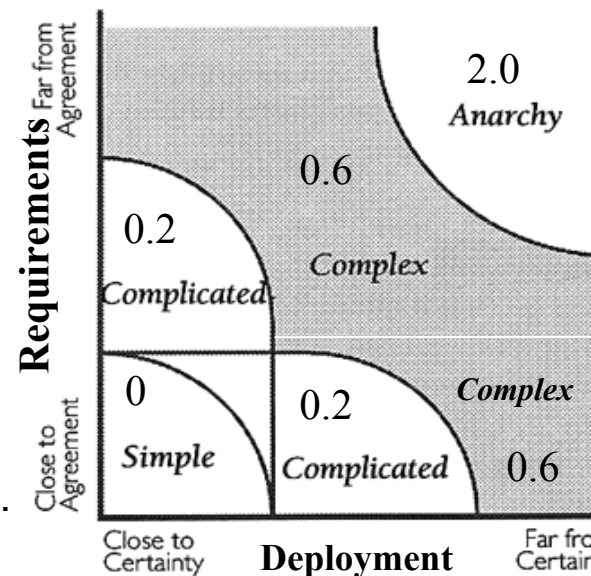
- Estimating is **iterative**. Estimates change as more information emerges.
- If you can't get a believable estimate for a top priority backlog item, **lower its priority** (to delay working on that item until you know more about it). Alternatively, keep them high priority but **reclassify** them **as an issue**. The work to turn this issue into required functionality might take ten days; so estimate the issue at ten days.
- Break down features to under 10 days** if they are to be implemented within the next 6 months.
- Lower priority product backlog is vague**, a placeholder for further investigation/thinking as its priority increases. The **estimates** (often 10-40 days) are **not very reliable**.

| Priorit y (1-9) | Function | Full Description | R D E |
|--------------------|---------------|--|-------------|
| 1 | General | Setup development environment | 4 |
| 1 | General | Confirm use of zope as development environment | 2 |
| 1 | Membership | Ability to sign up for various level of membership | 4 |
| 1 | Membership | Ability to use credit card to pay for membership | 13 |
| 1 | Membership | Provide extract from database to external sources | 3 |
| 1 | Membership | Notify members with membership data | 5 |
| 1 | Membership | Generate receipts and certificates | 5 |
| 1 | Membership | Tie membership program to bank accounts | 5 |
| 1 | Membership | Implement open access database (MySQL?) | 6 |
| 2 | News | Authoring environment for news | 8 |
| 1 | Website | Website look, feel, navigation, initial pages | 10 |
| 3 | Founders Page | Include agilealliance.org founders page | 10 |
| 3 | Sponsors | Display sponsors and link to web sites | 6 |
| 4 | Articles | Authoring environment for articles | 10 |
| 4 | Articles | Organizing and sorting capability for articles | 4 |
| 4 | Articles | Library catalog for articles | 8 |

From: Ken Schwaber: *Scrum Methodology*

Adjusting Release Backlog Estimates (1/2)

- To reflect any factors that reduce team effectiveness. Scrum assumes an optimal work environment. This activity **forces an understanding that suboptimal product factors have a cost.**
- The guidelines for adjusting estimates to reflect complexity are **just** that: **guidelines**. This is not a precise science.
- The estimated time for each item can be adjusted by a “**complexity factor**.” reflecting the degree of complexity due to **requirements and technology** that will affect the estimates.
- Keep adjustment for complexity **separate from the base estimate.**
- Apply** the complexity factor **only to a known horizon.** If it is possible that the team environment will change, don't apply complexity factors past that horizon.
- Diminish the impact of the complexity factors as the team can be expected to master the technical and business domain.



From: Ken Schwaber: *Scrum Methodology*

Adjusting Release Backlog Estimates (2/2)

- **Drag on team productivity:** when teams haven't worked together before, when they are not familiar with the technology, and when they aren't familiar with the business domain.
- **Adequacy of working environment:** when the team is not together in an open environment with adequate work and conference rooms. Usually a **factor of 0.2**
- **Multiple teams:** due to management and communications overhead caused by multiple teams. Usually an additional **factor of 0.1**
- **Adjusted Estimate = Raw Effort * (1 + complexity factor + drag + working environment + multiple teams)**

| Drag | # of years together | Knowledge of technology | Knowledge of domain |
|------|---------------------|-------------------------|---------------------|
| 0.8 | < 3 months | Low | Low |
| 0.75 | < 3 months | Low | Medium |
| 0.7 | < 3 months | Low | High |
| 0.75 | < 3 months | Medium | Low |
| 0.5 | < 3 months | Medium | Medium |
| 0.5 | < 3 months | Medium | High |
| 0.75 | < 3 months | High | Low |
| 0.5 | < 3 months | High | Medium |
| 0.35 | < 3 months | High | High |
| 0.6 | < 1 year | Low | Low |
| 0.55 | < 1 year | Low | Medium |
| 0.5 | < 1 year | Low | High |
| 0.55 | < 1 year | Medium | Low |
| 0.3 | < 1 year | Medium | Medium |
| 0.25 | < 1 year | Medium | High |
| 0.5 | < 1 year | High | Low |
| 0.25 | < 1 year | High | Medium |
| 0.2 | < 1 year | High | High |
| 0.5 | > 1 year | Low | Low |
| 0.45 | > 1 year | Low | Medium |
| 0.4 | > 1 year | Low | High |
| 0.45 | > 1 year | Medium | Low |
| 0.35 | > 1 year | Medium | Medium |
| 0.2 | > 1 year | Medium | High |
| 0.4 | > 1 year | High | Low |
| 0.2 | > 1 year | High | Medium |
| 0 | > 1 year | High | High |

From: Ken Schwaber: *Scrum Methodology*

User Stories (XP)

- A user story describes functionality that will be valuable to either a user or purchaser of a system or software. User stories are composed of three aspects.
 - A written description of the story used for planning and as a reminder (“**Card**”)
 - Conversations about the story that serve to flesh out the details of the story (“**Conversation**”)
 - Tests that convey and document details and that can be used to determine when a story is complete (“**Confirmation**”)
- While the Card may contain the text of the story, the details are worked out in the Conversation and recorded in the Confirmation.
- User stories can be **coded and tested between half a day and two weeks** by one or a pair of programmers.
- Because user stories **shift emphasis toward talking** and away from writing, important decisions are not captured in documents (that are unlikely to be read anyway). Instead, important aspects about stories are captured in automated acceptance tests and run frequently.
- User stories **encourage deferring detail** – it allows us to not spend time thinking about a new feature until we are sure that the feature is needed. Stories discourage us from pretending we can know and write everything down in advance.
- Try to implement at least **half a dozen of stories in an iteration**.

Mostly from: Mike Cohn: *User Stories Applied*

Estimating User Stories

- Size of story is given in “**story points**” (an abstract unit). The team defines how a story point translates to effort (typically: 1 story point = 1 ideal day of work).
- The number of story points that a team can deliver in an iteration is called “**team velocity**”.
- Estimate as a team
 - A **story comprises multiple tasks** which will be done by different people.
 - The estimate for the entire story is developed by the entire team, utilizing the experience of all members, similarly to the “Wideband Delphi” approach.
 - The **estimates are verified with triangulation and with previous estimates**.
- **Periodically re-estimate** a story, which gives you a chance to incorporate additional information.
- At the end of an iteration, **measure how much stuff got done**. Add up the ideal time in all the stories to determine the team’s velocity. Each developer measures his velocity by tracking his accomplishments every iteration. He can only sign up for the same number of day’s worth of tasks the next time (be careful not to use that data against him).

Mostly from: Mike Cohn: *User Stories Applied*

Adapted Delphi Wideband Approach

1. Gather together the customer and the developers. Bring along the story cards. Distribute a handful of blank cards to each participant.
2. The customer selects a story at random and reads it to the developers. The developers ask as many questions as they need.
3. When there are no more questions, each developer writes an estimate on a card, not yet showing the estimate to the others.
4. When everyone has finished, the estimators turn over their cards so everyone can see them.
5. If estimates differ, the high and low estimators explain their estimates.
6. The group discusses it for up to a few minutes, the customer clarifies issues. The developers estimate again write their estimates on cards. In many cases, the estimates will already converge by the second round. But, if they have not, repeat the process.
7. If the estimates differ slightly, reach group consensus on one value. Follow the rule "Optimism wins" (team members whose optimism burned the team once will learn to temper that optimism).

Mostly from: Mike Cohn: *User Stories Applied*

Triangulation

- After the first few estimates have been made, verify them by relating them to each other.
 - A two-point story should be half the effort of a four-point story.
 - A three-point story should be more roughly larger than a two-point story, but yet smaller than a four-point story.
- Although not exact, triangulation is an effective means for a team to verify that they aren't gradually altering the meaning of a story point. It builds on intuitive decision making.
- Pin the story cards to the wall based on their size, compare newly-estimated stories to others that may already have been implemented.
- Precision decreases as story size increases, therefore constrain estimates to pre-defined values (e.g. ½, 1, 2, 3, 5, 8, 13, 20, 40, 80)

| 1 | 2 | 3 | 5 | 8 | 13 |
|---------------|---------------|---------------|---------------|---------------|---------------|
| A user can... | A user can... | A user can... | A user can... | A user can... | A user can... |
| A user can... | A user can... | A user can... | A user can... | | A user can... |
| A user can... | | A user can... | A user can... | | |
| A user can... | | | A user can... | | |

From: Mike Cohn: *User Stories Applied*

Yesterday's weather

- Say you'll do as much today (in the next iteration) as you actually got done yesterday (in the last iteration).
- Emergent properties of this rule:
 - We won't habitually overestimate our abilities. We have to use actual accomplishment. If we overestimate once, we won't be able to the next time.
 - If we are overcommitted, we will tend to try to finish some items in any given time period instead of half finishing them all. It is so embarrassing to tell the customer they can have zero features next time.
 - On the other hand, if we have a disastrous period, we are guaranteed a little breathing space to recover.
 - Everyone learns to trust our numbers because they are so easy to explain.
 - Our estimates automatically track all kinds of changes – changes to the team, new technology, dramatic changes in product direction, etc.
- The first estimate (at the beginning of the project, when there is no yesterday) is the hardest and least accurate. Fortunately you only have to do it once.

From: Kent Beck and Martin Fowler: *Planning Extreme Programming*

Lean Software Development with 7 Principles and 22 Tools

- Eliminate Waste
 - Seeing Waste, Value Stream Mapping
- Amplify Learning
 - Feedback, Iterations, Synchronization, Set-Based Development
- Decide as Late as Possible
 - Options Thinking, The Last Responsible Moment, Making Decisions
- Deliver as Fast as Possible
 - Pull Systems, Queuing Theory, Cost of Delay
- Empower the Team
 - Self-Determination, Motivation, Leadership, Expertise
- Build Integrity In
 - Perceived Integrity, Conceptual Integrity, Refactoring, Testing
- See the Whole
 - Measurements, Contracts

From: Mary and Tom Poppendieck: *Lean Software Development*

Principles of agile estimation (1/2)

- Don't estimate waste, don't look too far into the future.

Estimate up to 2 iterations into the future (fine grain). Estimate up to 2 releases into the future (coarse grain).

- Estimate as a team (development + customer).

This reduces the “blame game”, you can no longer point accusing fingers at the person who made the plan.

- Estimate your own work.

You feel committed as an individual for the development of the self-selected tasks.

You feel committed as a group for delivering the iteration goal.

- Base estimates on facts, rather than on speculation. Use actual data. Use what happened in the past.

Yesterday's weather, team velocity, triangulation

- Estimate early.

Start getting estimates as soon as you write stories.

You will know what questions to ask if you can't estimate a story.

Eliminate Waste

Decide as Late as Possible

Amplify Learning

Empower the Team

Feedback

Measurements

Value-Stream Mapping

Amplify Learning

Principles

Tools

Principles of agile estimation (2/2)

- Estimate often. Learn from experience.
At the beginning of an iteration. Update the effort remaining daily / every other day.
- Estimate small features.
Stories of several days up to 2 weeks
- Keep it simple. Use simple rules.
- Communicate the constraints / assumptions of your estimates rather than just the numbers.
- Estimate total effort for implementing a story (development, testing, documentation etc.)
- Don't bring initial estimates (from release planning) into sessions for detailed estimates (iteration planning). Otherwise, the estimators will be tempted to force fit the task estimates to sum to the story estimate.
- Track the effort spent and the effort still open until completion. Don't ask for a percentage. Only count a story as implemented if it has passed the acceptance tests.

Amplify Learning
Empower the Team

Deliver as Fast as Possible

Making Decisions

Options Thinking

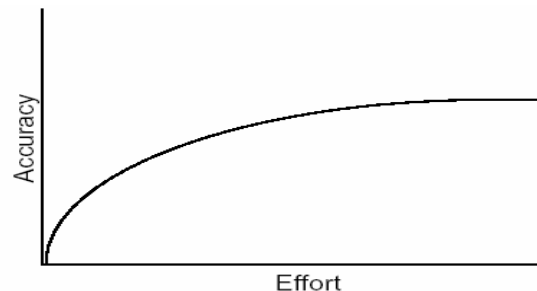
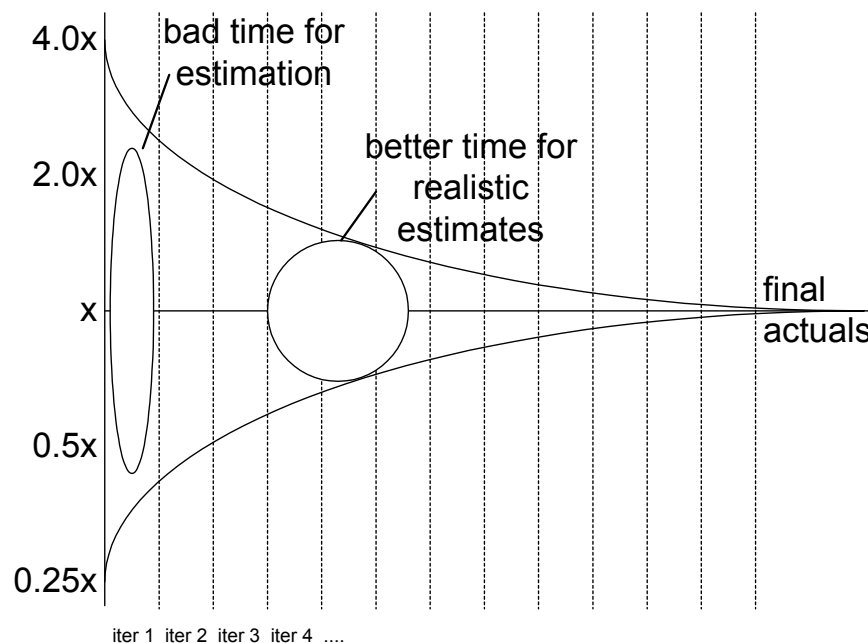
Measurements

Principles
Tools

Relationship to IBM Global Services Method

- There is a quite agile engagement model „Accelerated Development“ (formerly „Rapid Custom Development“).
- Every (?) engagement model contains at the beginning of each phase the task „Refine Project Estimates“, so iterative estimation is not a new concept.
- The main switch is from specialist estimation to group estimation.
- Emergence of estimates might be hard to accept, but the quality of estimates will always have to be commensurate with the information available.
- PMI has the notion of “rolling wave planning” which is very similar to the adaptive estimation approach.
- Put just enough effort into estimation, and be aware of the diminishing returns, when additional effort doesn't result into a much more accurate estimate.

Cone of Uncertainty



Summary

- In a complex environment with frequent changes that cannot be anticipated, estimation must be done in an incremental and adaptive way. It can no longer be done in a sophisticated, big-upfront approach by highly skilled estimators at the beginning of the project.
- Agile approaches to estimation look extremely simple, but they work (somehow unexpectedly) quite well.
- Analogies to Lean Software Development (and Lean Manufacturing) explain why these simple approaches work.
- If you want to apply different (traditional, more complex, more rigorous,...) approaches in a complex environment, make sure that you understand the principles of Lean Software Development to check whether the approach can possibly work.

Questions?





IBM Global Services

Backup Slides

Next to estimation

- Estimating bugs
- Iceberg list (Crystal)
- Blitz planning (Crystal)
- Planning Game (XP)
- Sprint Planning Meeting (Scrum)
- Burn-down Chart (Scrum)
- Spatial Reasoning
- Wideband Delphi

Estimating Bugs

- First, determine if the bug is critical (= can't wait until the next iteration).
 - Only the customer can decide whether the bug is really critical.
 - Make the tradeoff “bug vs. function” explicit, since a fixed bug might push a story into the next release (see the “Iceberg list”).
- If it's not critical, then log it on a card. Get development to look at it and estimate the effort. You can mark the effort as unknown. If it's less than an ideal day, mark it as small.
- If the estimate is more than a day's worth of effort, treat the defect as a story. The customer can then say which iteration should address the defect, just as with any other story. Usually it's worth lumping several bugs together to get a week's worth.
- Just before the next iteration planning meeting, the customer should take the small and unknown bugs and prioritize them. The customer should indicate how much ideal time the developers should spend dealing with them.
- Encourage everyone to deal with bugs in a rational way, to make sensible tradeoffs between fixing defects and adding features.

From: Kent Beck and Martin Fowler: *Planning Extreme Programming*

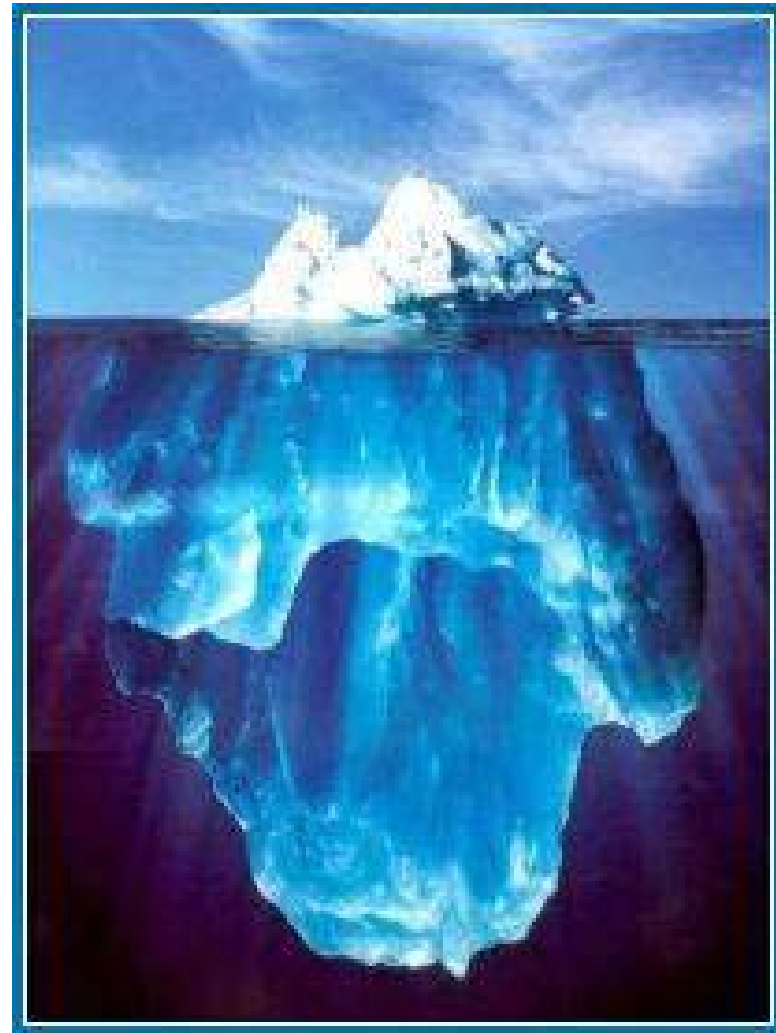
Iceberg List (Crystal, Scrum)

- The “above water” part lists all the items that can be delivered in the current delivery cycle. The “below water” part lists every item that will be delivered in later cycles.
- When you add a new item to the above water part, it pushes everything else down, and something that was above water falls below water.
- The iceberg list is useful in hostile environments, where sponsors change the requirements and priorities on a daily basis.

- Product Backlog (Release Backlog):
 - List of functionality, technology, issues (placeholders that are later defined as work)
 - Emergent, prioritized, estimated with more detail on higher priority backlog
 - Product Owner responsible for priority, anyone can contribute
 - Maintained and posted visibly
- Sprint Backlog (Iteration Backlog):
 - Product Backlog selected for Sprint by team, team selects tasks to turn product backlog into working product functionality.
 - Cannot be added to or changed during Sprint from the outside, only team member can add, delete or change the Sprint Backlog.

Sprint Backlog

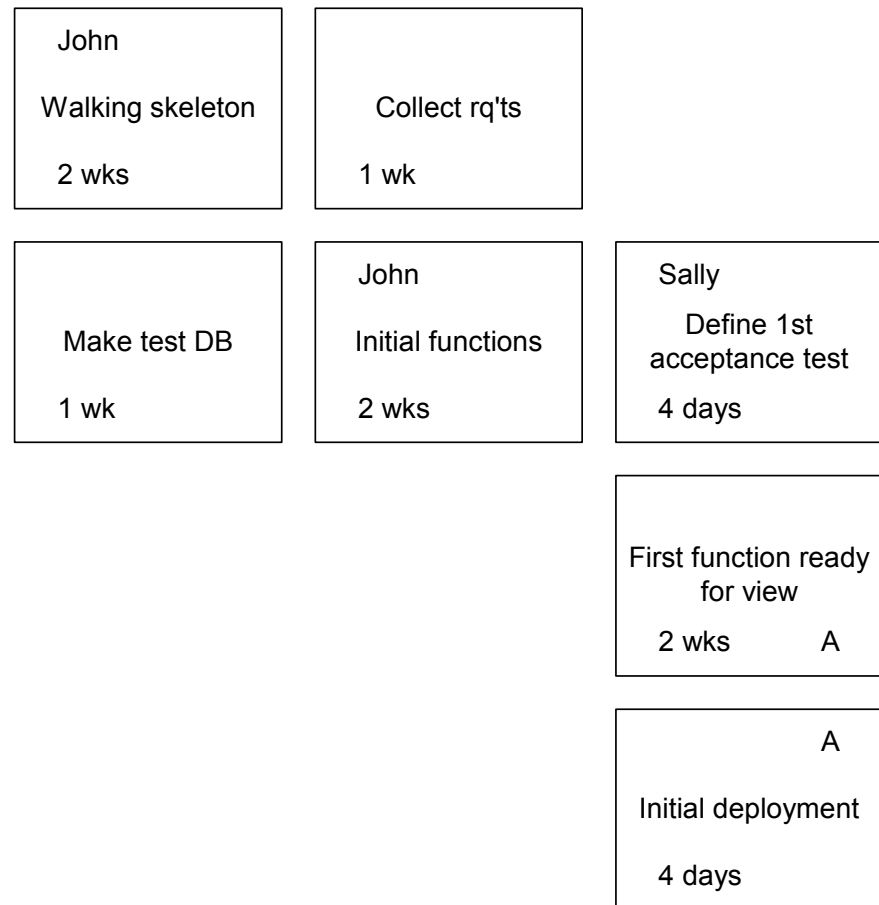
Product Backlog



From: Alistair Cockburn: *Crystal Clear*

Blitz Planning (Crystal)

1. Gather the attendees (representatives from each stakeholder category)
2. Brainstorm the tasks (5-15 minutes)
3. Lay out the tasks on a big table in dependency order (top to bottom), parallel tasks next to each other, remove duplicates
4. Review the tasks, add tasks as needed
5. Estimate effort and tag the tasks with names of specific people required. Identify over-loaded people.
6. Sort the tasks (parallel / sequential)
7. Mark the walking skeleton, the earliest release and the earliest revenue
8. Identify other releases
9. Optimize the plan to fit the project priorities, re-prioritize, re-sort, delay tasks
10. Capture the output and display publicly



From: Alistair Cockburn: *Crystal Clear*

Planning Game (XP)

- People place index cards on the table, one user story per card.
- The group pretends there are no dependencies between cards, and simply lines them up in the preferred sequence of development
- The developers write on each card an estimate of how long it will take to produce the function.
- The sponsor or expert user puts them into development priority sequence, taking into account the development time and business value of each function.
- The cards are clustered into (fixed-length) iterations, and those iterations are clustered into releases, usually not longer than a few months each.

From: Kent Beck and Martin Fowler: *Planning Extreme Programming*

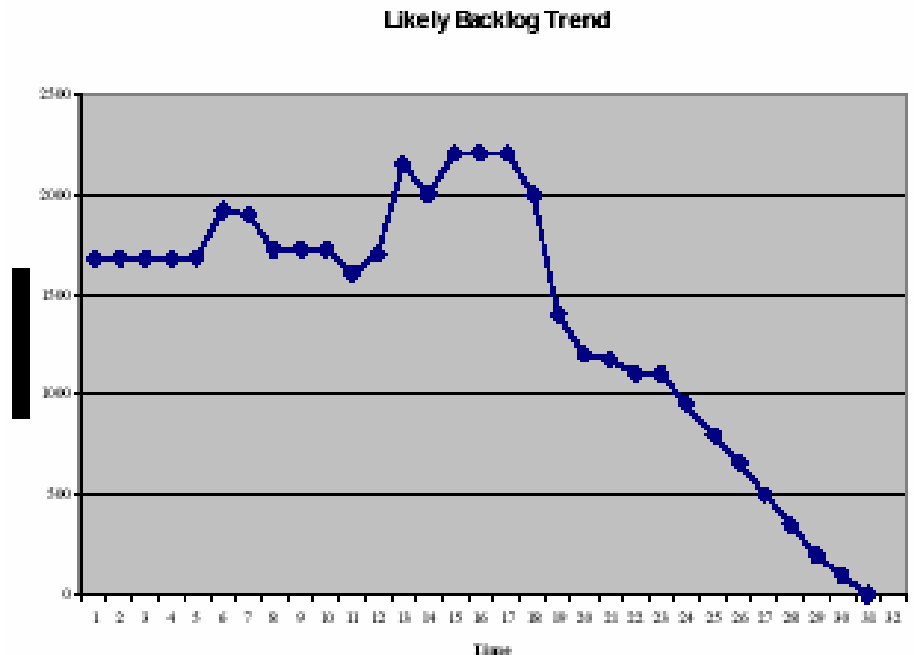
Sprint Planning Meeting (Scrum)

- 4 hours (max) for team to select Product Backlog and set Sprint Goal with Product Owner
 - Attended by Product Owner, Scrum team, customers, management in order to define what to build in the next Sprint
 - Team selects as much Product Backlog as it believes it can develop during the next Sprint.
 - Product Owner and customer devise Sprint Goal (which is the business value that the product increment must deliver regardless of functionality implemented).
- 4 hours (max) for team to define Sprint Backlog
 - Attended by Product Owner, Scrum team, development management in order to define how to build the product functionality into a product increment in the next Sprint.
 - Team defines Sprint Backlog, consisting of all tasks that need to be completed during Sprint.
 - Team members sign up for work and estimate their tasks.
 - Tasks are 1-16 hours long (XP suggests 1-3 days); if longer, break them down into more granularity.

From: Mike Beedle and Ken Schwaber: *Agile Software Development with Scrum*

Burn-down Chart (Scrum)

- Burn-down Chart shows the estimated number of hours required to complete the tasks of the Sprint.
- It shows both the status and rate of progress (“velocity”) in a way that is both clear and easy to discuss.
- Posted visibly.
- Similar to earned-value chart if you count the delivered functionality (instead of development effort) over time.



From: Mike Beedle and Ken Schwaber: *Agile Software Development with Scrum*

Spatial Reasoning

- Create a game table with enough slots for the stories of an iteration.
- Create physical cards for the stories where the size of the card matches the story points.
- Validate whether the size of the cards feel right.
- Decide how to organize the story cards on the game board.
- By sizing the pieces so that all math can be done quickly using visual inspection, the planning meeting remains focused on the competing business issues.

available slots

| | | |
|--|--|--|
| | | |
| | | |
| | | |
| | | |

actual stories

| | |
|------------------------------------|---------------------------------|
| Bug 1 pt. | Big user story 3 pts. |
| Typical user story 2 pts. | |

Wideband Delphi

1. Kickoff Meeting: At least three people discuss the source documents and project, as well as the units of estimation.
2. Estimation: Each person creates three estimates (using his/her preferred method): most likely, optimistic, pessimistic case.
3. Meeting: Each estimator gives his/her estimates to the facilitator who displays them with averages (owners of the estimates may not be revealed if it's desired to reduce the influence of personality or seniority). Each estimator discusses the insights, problems and assumptions.
4. Repeat steps 2 and 3 at least once to get iterative estimation refinement: let the feedback drive adaptation and improvement
5. Calculate the final numbers using the averages from the final cycle.

$$\text{Estimate} = (\text{Optimistic} + \text{Pessimistic} + 4 * \text{Most likely}) / 6$$

$$\text{Likely Deviation} = (\text{Pessimistic} - \text{Optimistic}) / 6$$

Wideband Delphi sits on top of any other estimation method, improving it through multiple participants, feedback, and iterative refinement

References

- Kent Beck and Martin Fowler: *Planning Extreme Programming*, Addison-Wesley, 2001.
- Mike Beedle and Ken Schwaber: *Agile Software Development with Scrum*, Prentice Hall, 2001.
- Alistair Cockburn: *Crystal Clear*, Addison-Wesley, 2005.
- Mike Cohn: *User Stories Applied*, Addison-Wesley, 2004.
- Mike Cohn: *Agile Estimation and Planning*, to be published, work in progress is available online at <http://www.mountangoatsoftware.com/agileplanning/>
- George D. Githens: *Rolling Wave Project Planning*,
<http://www.catalystpm.com/NP02.PDF>
- Craig Larman: *Agile & Iterative Development*, Addison-Wesley, 2004.
- Todd Little: *Agility, Uncertainty, and Software Project Estimation*
<http://www.macs.ece.mcgill.ca/~radu/304428W03/AgilityUncertaintyAndEstimation.pdf>
- Ogunnaike and Ray: *Process Dynamics, Modeling, and Control*, Oxford University Press, 1992.
- Mary and Tom Poppendieck: *Lean Software Development*, Addison-Wesley, 2003.
- Ken Schwaber: *Agile Project Management with Scrum*, Microsoft Press, 2004.
- Ken Schwaber: *Scrum Methodology Revision 0.9*, Advanced Development Methods Inc., 2003.