ITSS SOFTWARE DEVELOPMENT

# 4. AGILE SOFTWARE DEVELOPMENT

# Content

1. Non-agile Software Processes
2. Agile Software Development
3. eXtreme Programming (XP)
4. Scrum

# Content

1. Non-agile Software Processes
2. Agile Software Development
3. eXtreme Programming (XP)
4. Scrum

# Traditional Software Development Process


- **Recognition that:**
  - Highly defined process with predictable start and finish dates for tasks is unrealistic
  - Predictive, phased-project waterfall approach not working
- **Reality is:**
  - Requirements change
  - SW development is **intellectually intensive**, **creative** process
  - Often developing in **highly complex** and **uncertain domain**
  - Requires **empirical** management & control process – inspect and adapt feedback loops

*Moving toward*

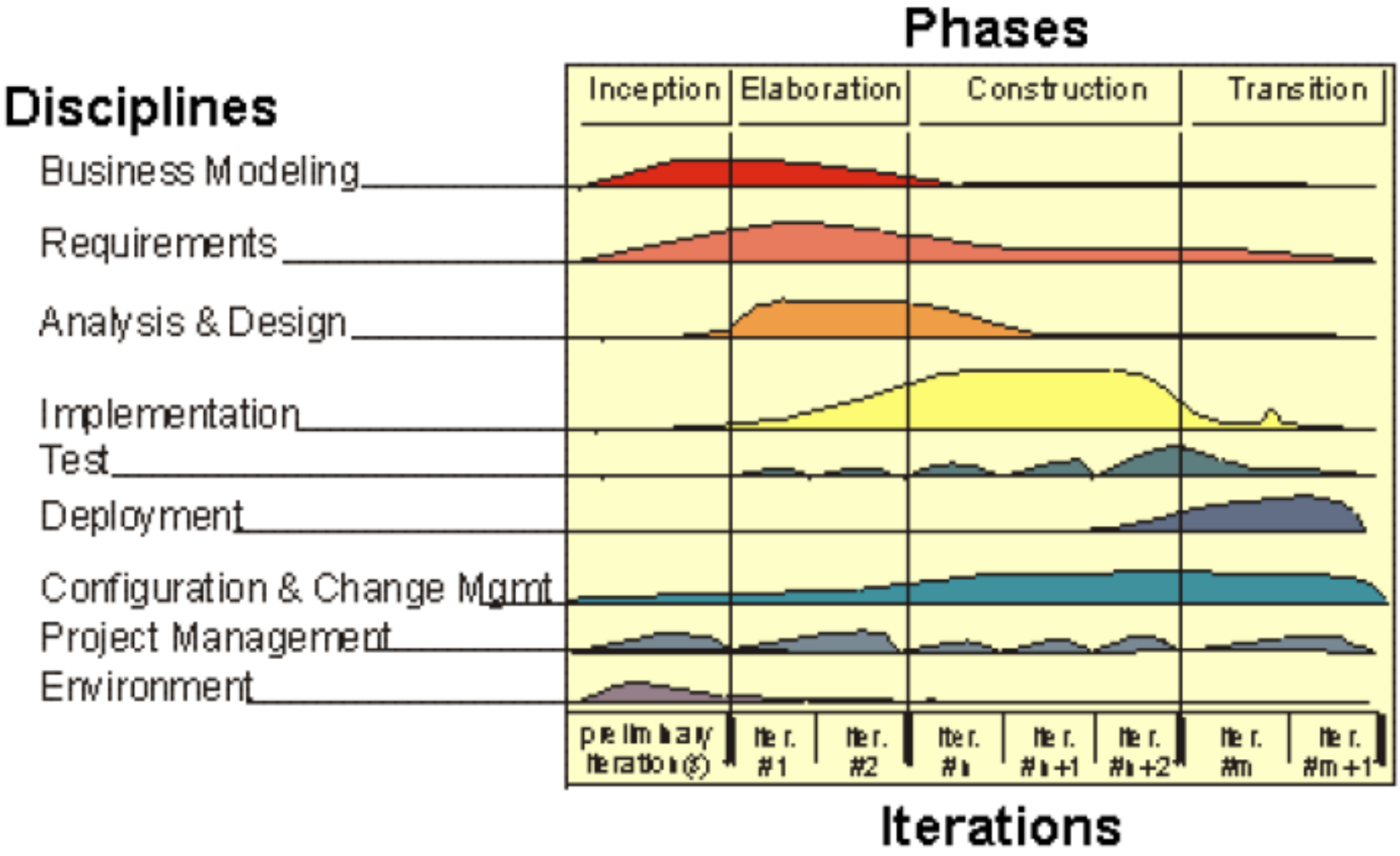
*Iterative* & *Incremental*

*Software Development Processes*

# The Unified Process (UP)

# Ration Unified Process (RUP) – Video

# Unified Process Lifecycle



# Unified Process

- RUP
  - An instance of the Unified Process
  - The most famous **heavyweight** Process
- Makes the most use of UML Diagrams (modeling)
- The main problem with RUP is its **slowness in reaching the code**, due to **high emphasis on documentation and modeling**
- In software development with RUP, the prescribed process is highly detailed and has significant important
  - Benefits ?
  - Drawbacks?

*Moving toward*

***Lightweight***

*Software Development Processes*

# Agile Software Development

# Content

1. Non-agile Software Processes
2. Agile Software Development
3. eXtreme Programming (XP)
4. Scrum

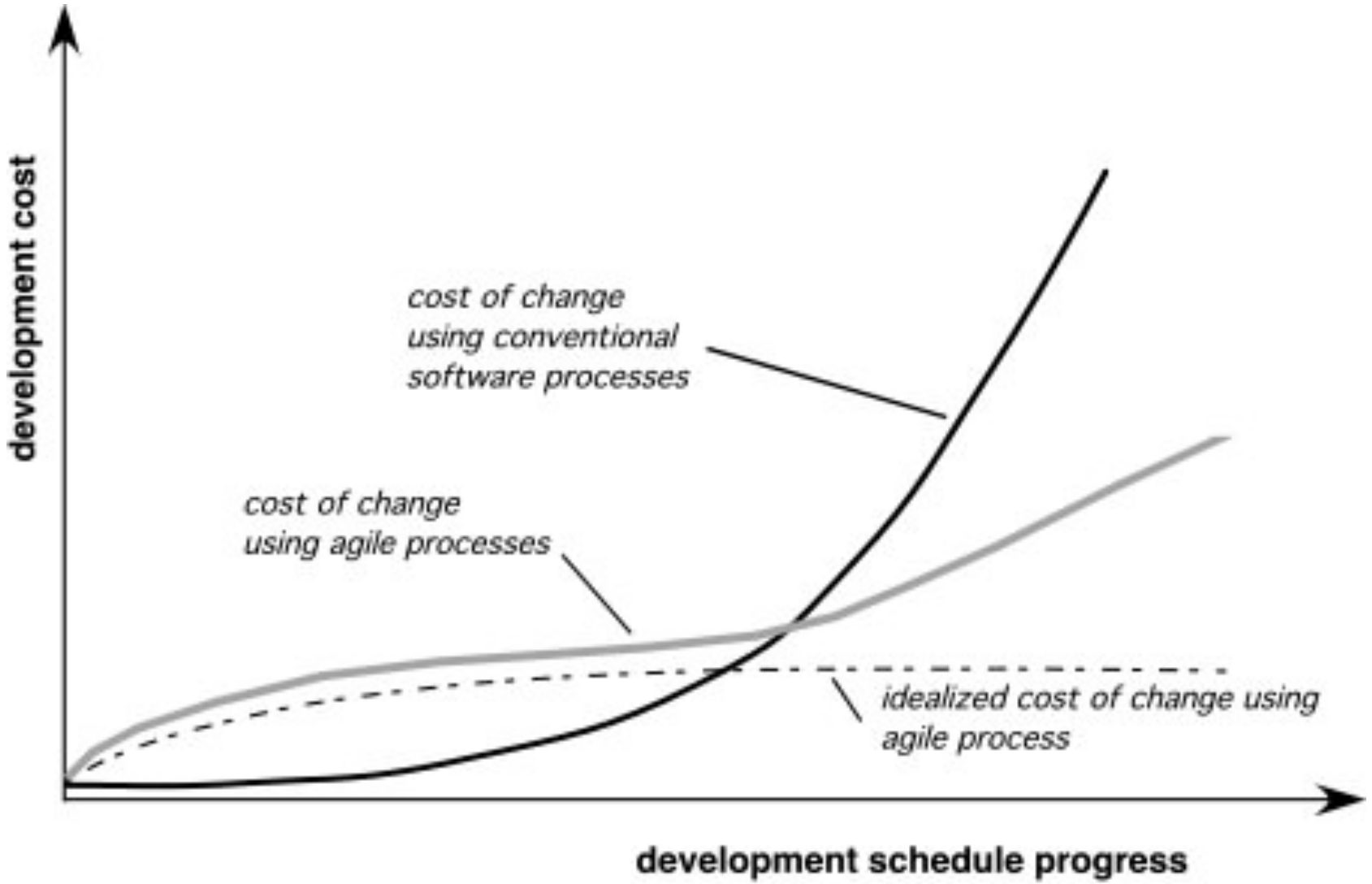
# What is "Agility"?

- Effective (rapid and adaptive) **response to change**
- Effective **communication** among all stakeholders
- Drawing the **customer** onto the team
- Organizing a **team** so that it is in control of the work performed

*Yielding …*

- **Rapid, incremental delivery of software**

# Agility and the Cost of Change



# History of Agile Software Development (SD)

- Incremental SD: 1957
- Adaptive SE process: 1974 (E. A. Edmond)
- Evolution of agile SD: Mid 1990s
  - Reaction to more heavyweight, document-driven methods (waterfall & RUP basically).
- In February 2001, 17 developers met in Snowbird, Utah, to discuss **lightweight** SD and published the Agile Manifesto.
  - This signaled industry acceptance of agile philosophy.

# Agile Manifesto

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

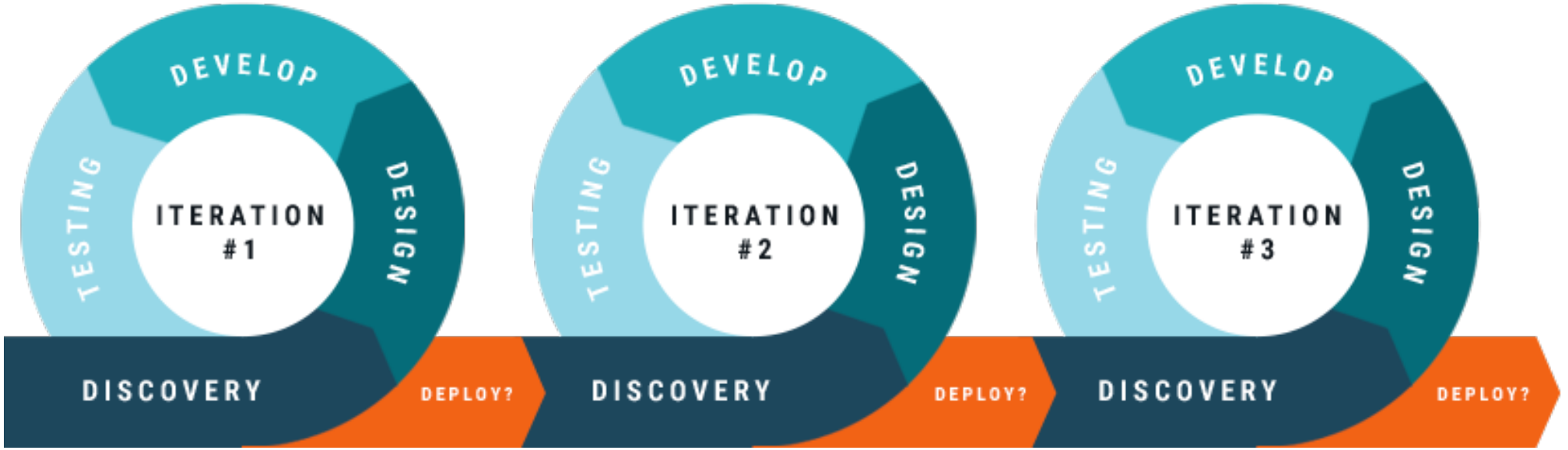
- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

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

*Kent Beck et al*

# An Agile Process

- Driven by customer descriptions of what is required
- Recognizes that plans are short-lived
- Develops software iteratively with a heavy emphasis on construction activities
- Delivers multiple “software increments”
- Adapts as changes occur

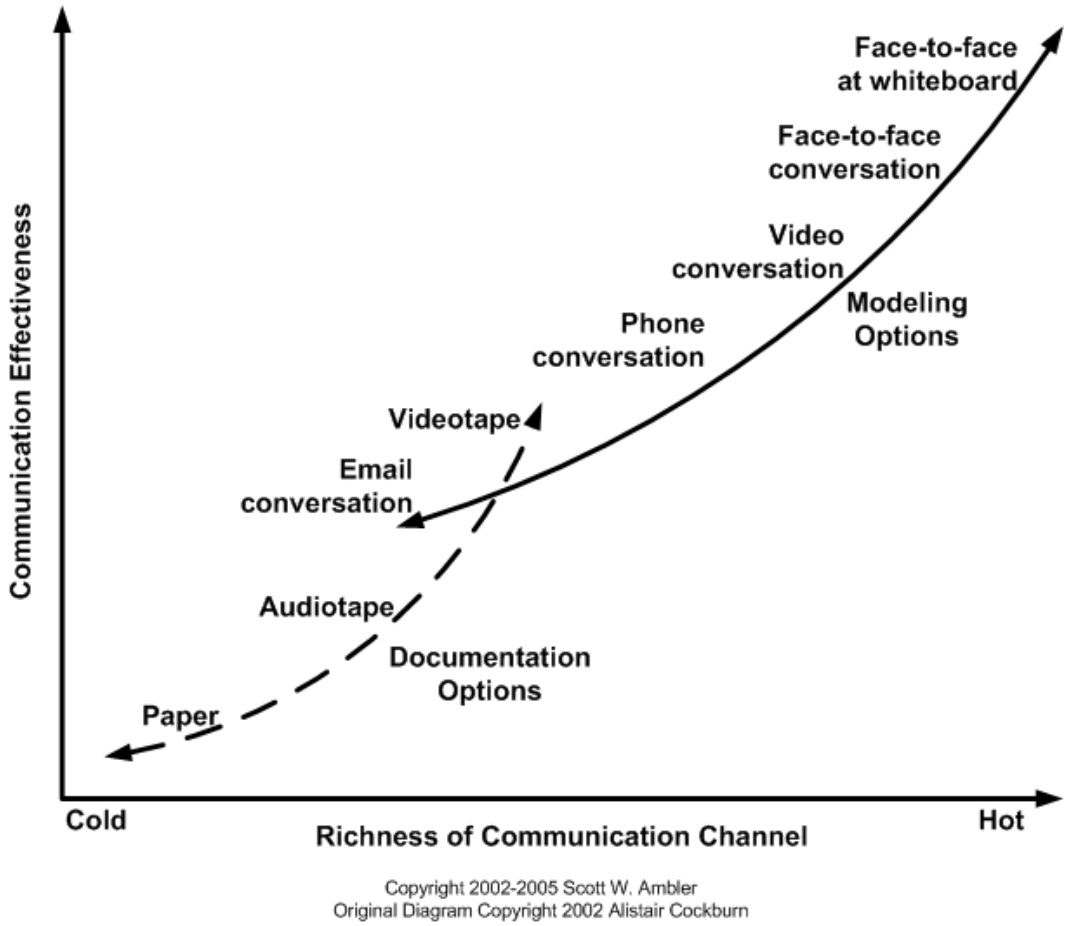


# Twelve Agile Principles

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**

# Twelve Agile Principles

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.






- **Try to follow the most effective communication technique applicable to your situation**
- **Be prepared to change your approach throughout a project**

# Twelve Agile Principles

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.

# Human Factors

- *The process molds to the needs of the people and team,* not the other way around
- Key traits must exist among the people on an agile team and the team itself:
  - **Competence**
  - **Common focus**
  - **Collaboration**
  - **Decision-making ability**
  - **Fuzzy problem-solving ability**
  - **Mutual trust and respect**
  - **Self-organization.**

# Who should / should not use Agile?

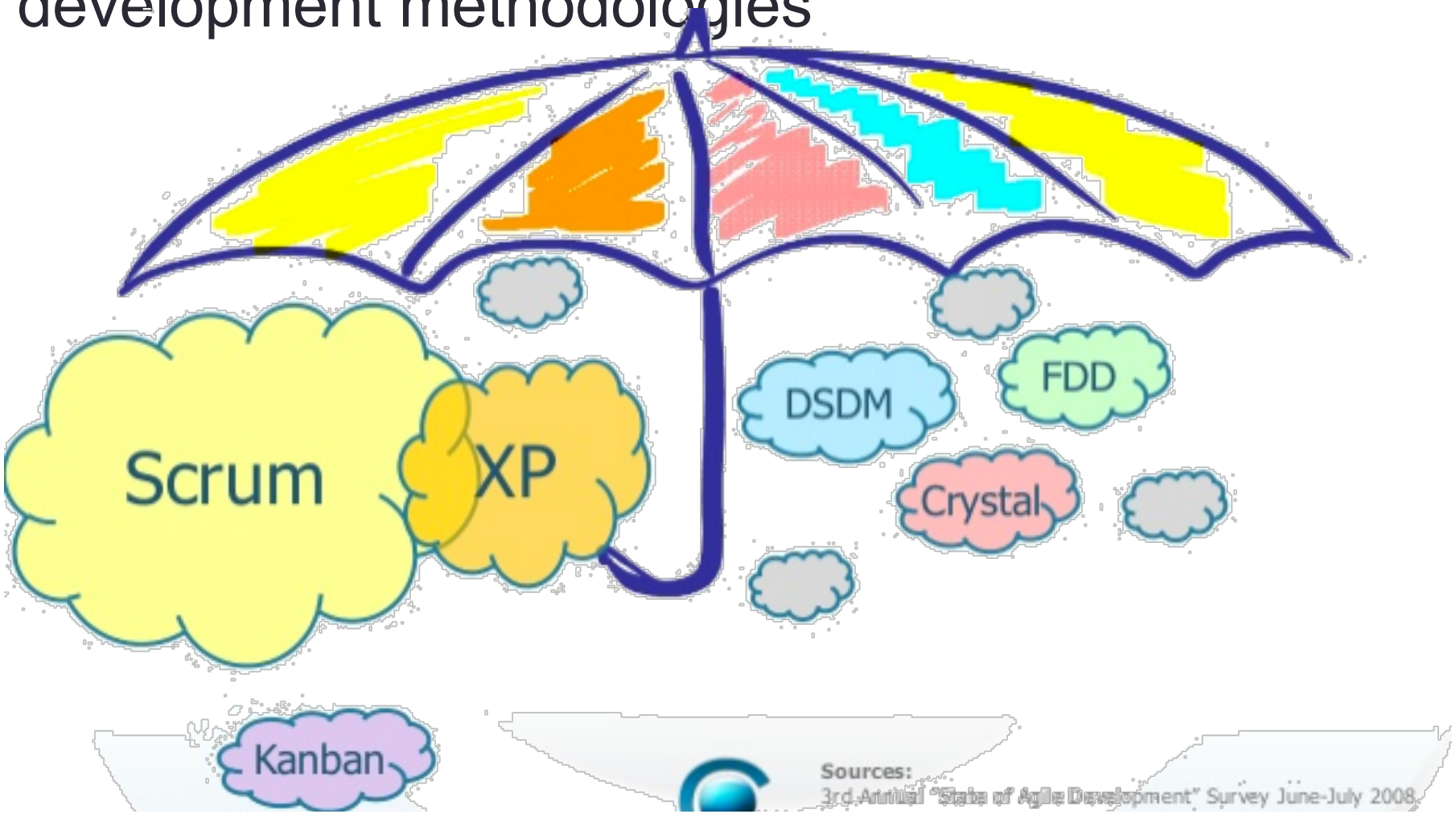
- Agile home ground:
  - Low criticality
  - Senior developers??
  - Requirements change often
  - Small number of developers
  - Culture that thrives on chaos

- Plan-driven home ground:
  - High criticality
  - Junior developers ??
  - Requirements do not change often
  - Large number of developers
  - Culture that demands order

- Formal methods:
  - Extreme criticality
  - Extreme quality
  - Senior developers
  - Limited requirements
  - limited features

# Agile umbrella

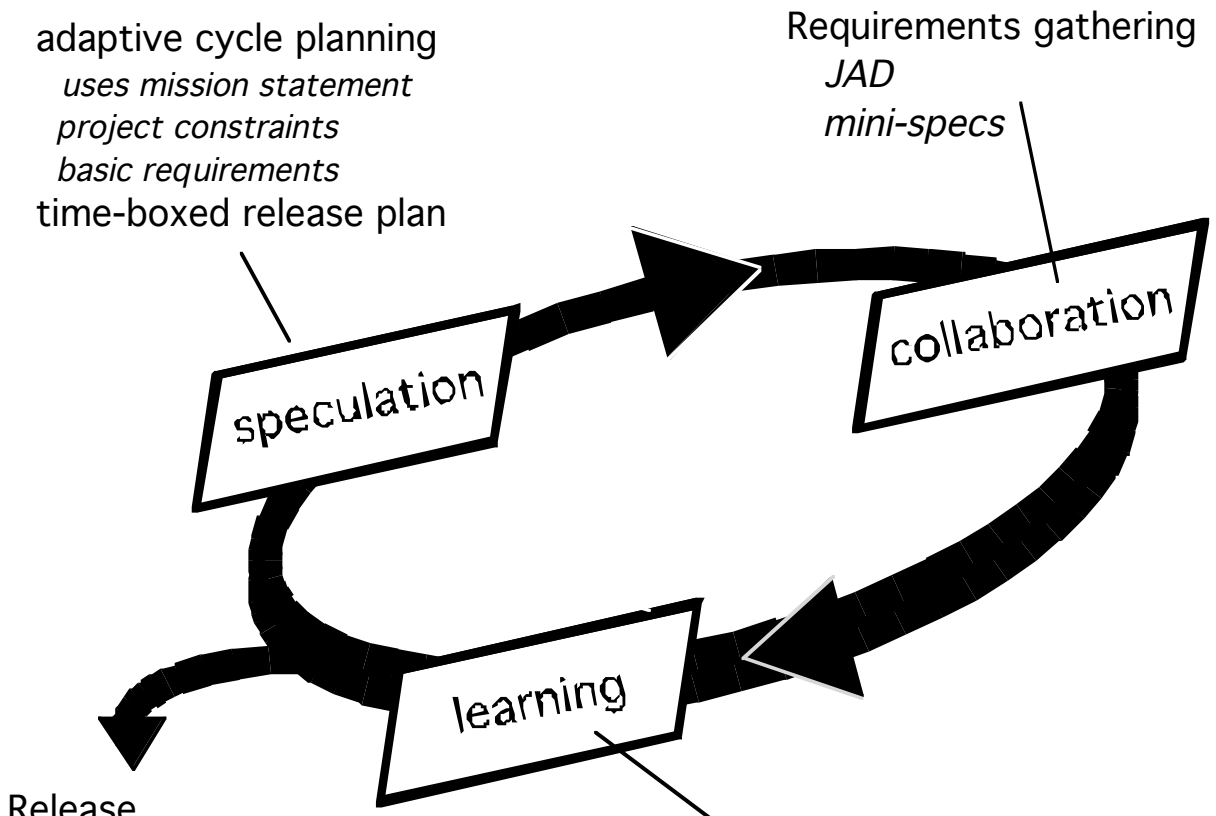
- “Agile Development” is an umbrella term for several iterative and incremental software development methodologies



# Adaptive Software Development

- Originally proposed by Jim Highsmith
- ASD — distinguishing features
  - Mission-driven planning
  - Component-based focus
  - Uses "time-boxing" (See Chapter 24)
  - Explicit consideration of risks
  - Emphasizes collaboration for requirements gathering
  - Emphasizes "learning" throughout the process

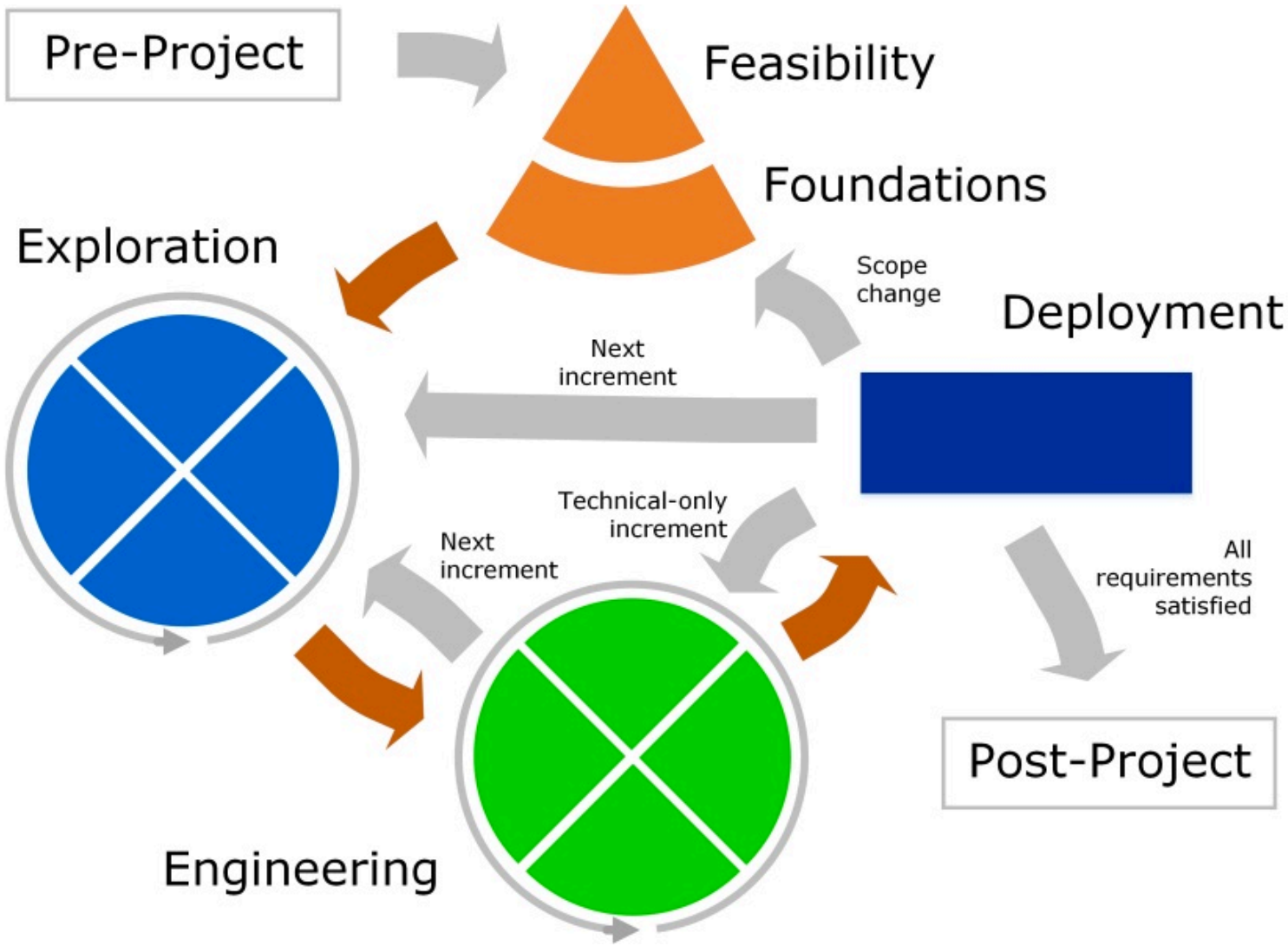
# Adaptive Software Development



# Dynamic Systems Development Method

- Promoted by the DSDM Consortium (www.dsdm.org)
- DSDM—distinguishing features
  - Similar in most respects to XP and/or ASD
  - Nine guiding principles
    - Active user involvement is imperative.
    - DSDM teams must be empowered to make decisions.
    - The focus is on frequent delivery of products.
    - Fitness for business purpose is the essential criterion for acceptance of deliverables.
    - Iterative and incremental development is necessary to converge on an accurate business solution.
    - All changes during development are reversible.
    - Requirements are baselined at a high level
    - Testing is integrated throughout the life-cycle.

# Dynamic Systems Development Method



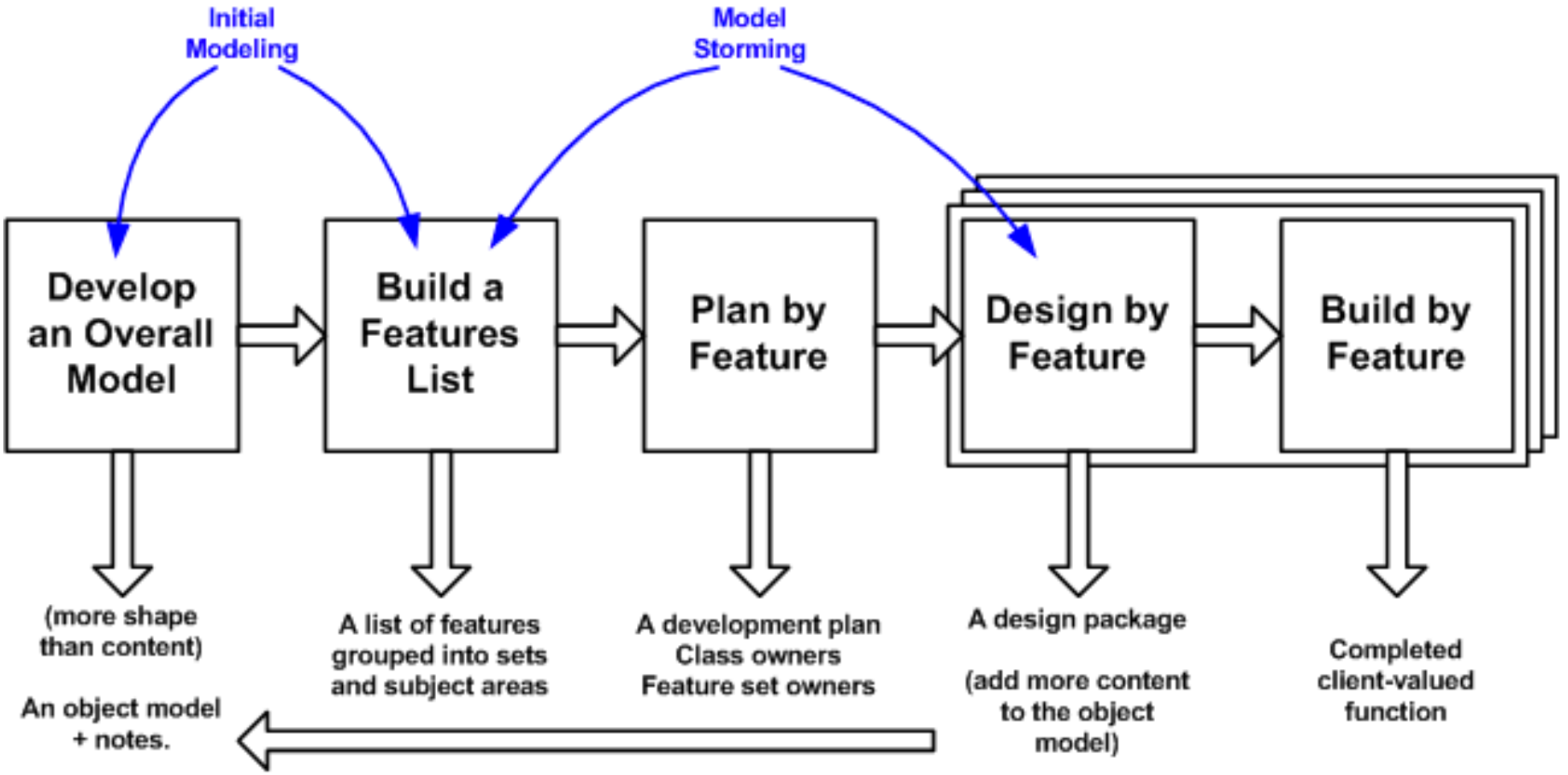
# Crystal

- Proposed by Cockburn and Highsmith
- Crystal—distinguishing features
  - Actually a family of process models that allow "maneuverability" based on problem characteristics
  - Face-to-face communication is emphasized
  - Suggests the use of "reflection workshops" to review the work habits of the team

# Feature Driven Development

- Originally proposed by Peter Coad et al
- FDD—distinguishing features
  - Emphasis is on defining "features"
    - a *feature* "is a client-valued function that can be implemented in two weeks or less."
  - Uses a feature template
    - <action> the <result> <by | for | of | to> a(n) <object>
  - A features list is created and "plan by feature" is conducted
  - Design and construction merge in FDD

# Feature Driven Development



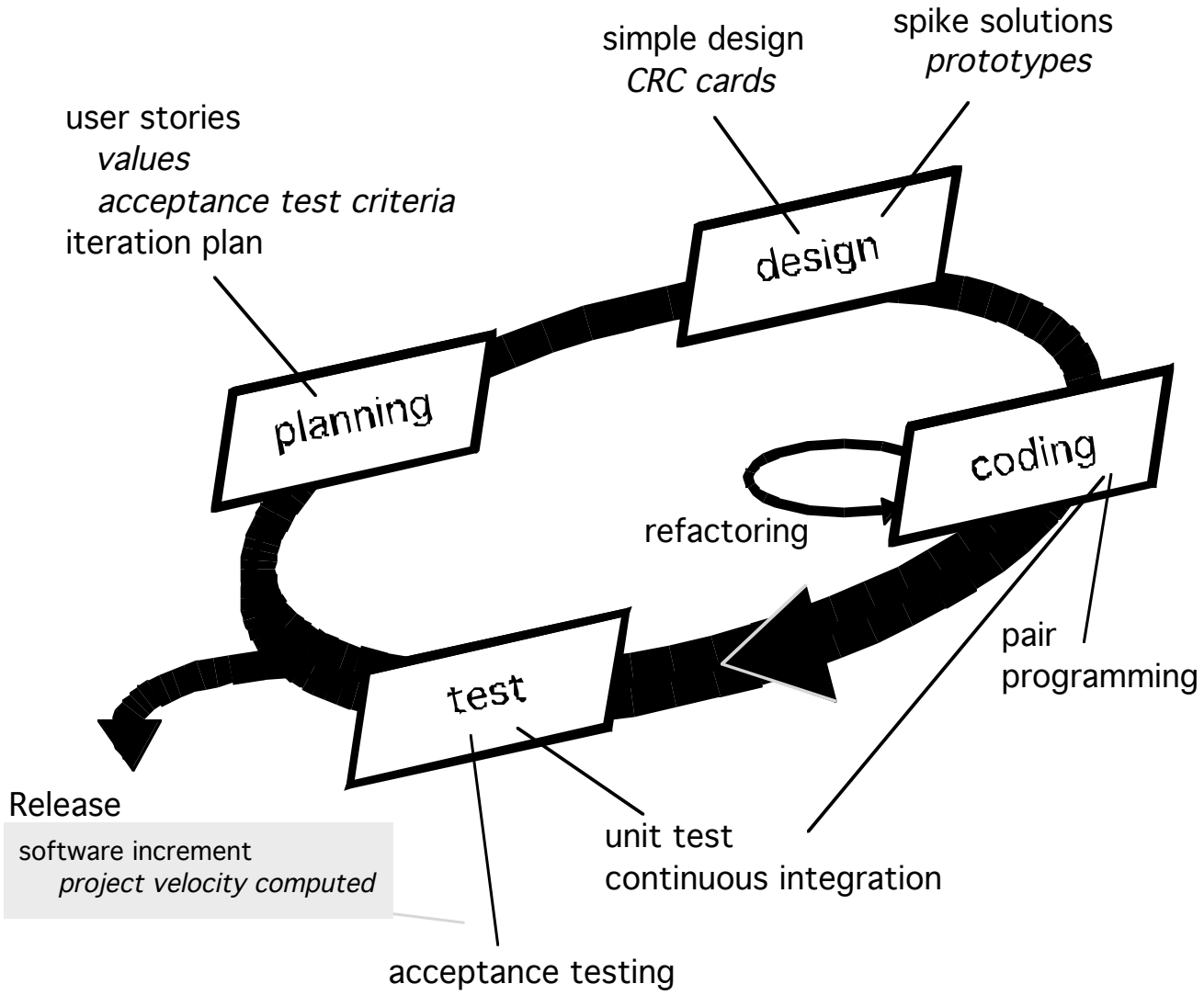
# Content

1. Non-agile Software Processes
2. Agile Software Development
3. eXtreme Programming (XP)
4. Scrum

# Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck, refined version in 2004
- Improves software quality and responsiveness to changes
  - Short development cycles (time-boxing), less loss
- Avoids features until they are actually needed
- XP Planning
  - Begins with the creation of "user stories"
  - Agile team assesses each story and assigns a cost
  - Stories are grouped to for a deliverable increment
  - A commitment is made on delivery date
  - After the first increment, "project velocity" is used to help define subsequent delivery dates for other increments

# Extreme Programming (XP)



# Extreme Programming (XP)

- XP Design
  - Follows the KIS principle
  - Encourage the use of CRC cards
  - For difficult design problems, suggests the creation of "spike solutions" — a design prototype
  - Encourages "refactoring"—an iterative refinement of the internal program design
- XP Coding
  - Recommends the construction of a unit test for a store *before* coding commences
  - Encourages "pair programming"
- XP Testing
  - All unit tests are executed daily
  - "Acceptance tests" are defined by the customer and excuted to assess customer visible functionality
  - "Testathon" – collaborative test writing

# XP Values

- **Communication**
  - Rapidly building and disseminating institutional knowledge among members of a development team
  - The goal is to give all developers a shared view of the system which matches the view held by the users of the system
- **Simplicity**
  - Extreme Programming encourages starting with the simplest solution. Extra functionality can then be added later.
  - “You ain't gonna need it”
- **Feedback**
  - Feedback from the system: by writing unit tesrt or running periodic integration tests
  - Feedback from the customer: The functional tests (aka acceptance tests) are written by the customer and the testers
  - Feedback from the team: planning game (for new ) and retrospectives
- **Courage**
  - One is the commandment to always design and code for today and not for tomorrow [Refactoring will improve it]
  - knowing when to throw code away: courage to remove source code that is obsolete, no matter how much effort was used to create that source code
  - persistence: A programmer might be stuck on a complex problem for an entire day, then solve the problem quickly the next day, if only they are persistent.
- **Respect**
  - For others as well as self-respect
  - never commit changes that break compilation, that make existing unit-tests fail, or that otherwise delay the work of their peers.
  - always striving for high quality and seeking for the best design for the solution at hand through refactoring

# XP Practices

- **Fine scale feedback**
  - **Pair-programming** -- all production code is written with two programmers at one machine
  - **Planning game** – determine scope of the next release by combining business priorities and technical estimates
  - **Test-Driven Development** – programmers continuously write unit tests; customers write tests for features
  - **On-site customer** – a user is on the team and available full-time to answer questions
- **Continuous process**
  - **Continuous integration** – integrate and build the system many times a day – every time a task is completed.
  - **Refactoring** – programmers continuously restructure the system without changing its behavior to remove duplication and simplify
  - **Small releases** – put a simple system into production, then release new versions in very short cycle

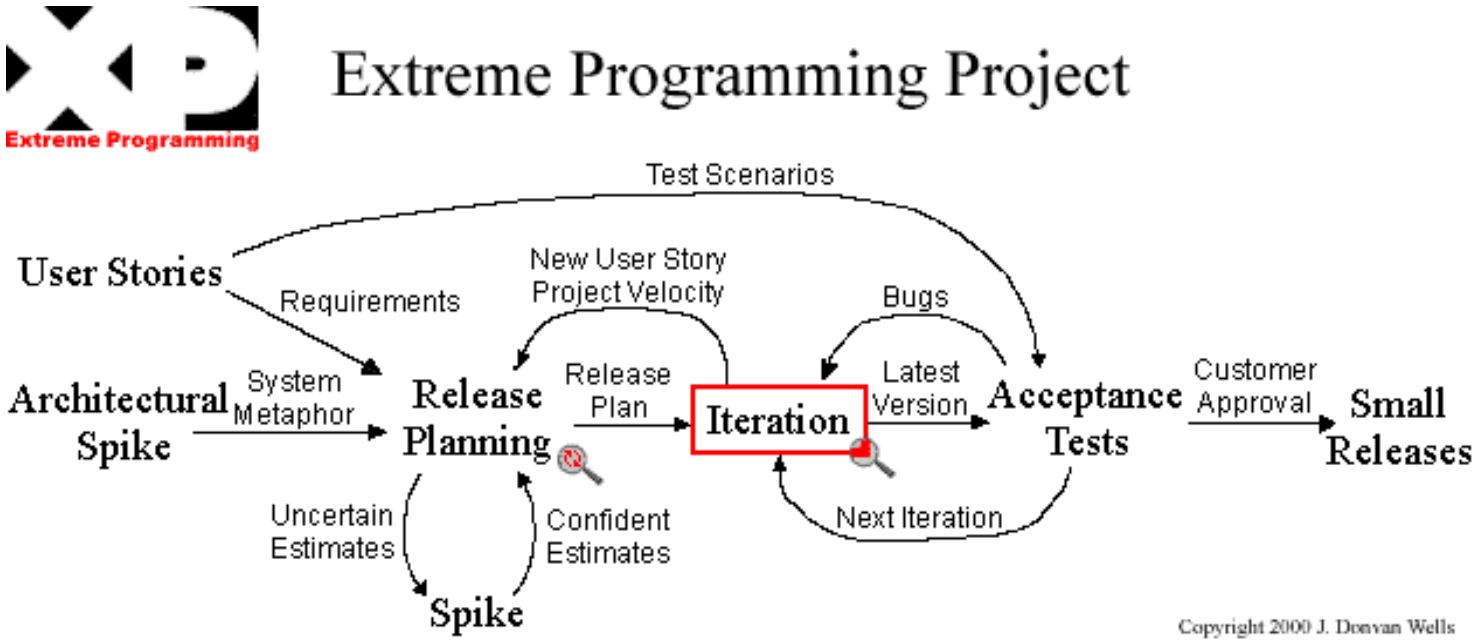
# XP Practices

- **Shared understanding**
  - **Coding standards** – programmers write all code in accordance with rules emphasizing communication through the code
  - **Collective ownership** – anyone can change any code anywhere in the system at any time.
  - **Simple design** – system is designed as simply as possible (extra complexity removed as soon as found)
  - **System Metaphor** – all development is guided by a simple shared story of how the whole system works
- **Programmer welfare**
  - **40-hour week** – work no more than 40 hours a week as a rule

# XP Values

- **Communication**
  - Rapidly building and disseminating institutional knowledge among members of a development team
  - The goal is to give all developers a shared view of the system which matches the view held by the users of the system
- **Simplicity**
  - Extreme Programming encourages starting with the simplest solution. Extra functionality can then be added later.
  - “You ain't gonna need it”
- **Feedback**
  - Feedback from the system: by writing unit tests or running periodic integration tests
  - Feedback from the customer: The functional tests (aka acceptance tests) are written by the customer and the testers
  - Feedback from the team: planning game (for new ) and retrospectives
- **Courage**
  - One is the commandment to always design and code for today and not for tomorrow [Refactoring will improve it]
  - knowing when to throw code away: courage to remove source code that is obsolete, no matter how much effort was used to create that source code
  - persistence: A programmer might be stuck on a complex problem for an entire day, then solve the problem quickly the next day, if only they are persistent.
- **Respect**
  - For others as well as self-respect
  - never commit changes that break compilation, that make existing unit-tests fail, or that otherwise delay the work of their peers.
  - always striving for high quality and seeking for the best design for the solution at hand through refactoring

# XP Process



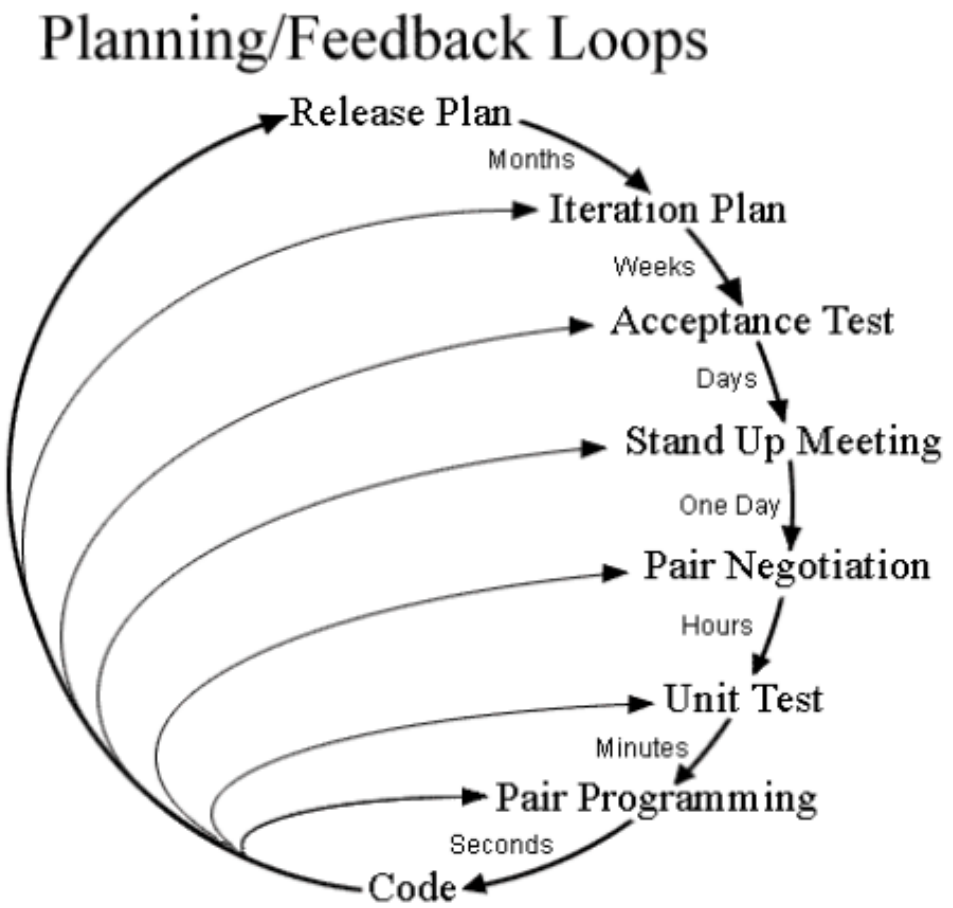

User Story:

- As a <role>, I want <goal/desire> so that <benefit>
- As a <role>, I want <goal/desire>

* Should be written by the customers
* It is informal and does not contain technical details, also it represents WHAT, not HOW
* Should not be long (often in one to three sentences – fit on 3x5 inches cards )
* Before implementation every US should be completed by relevant Acceptance tests

# XP Values

- **Communication**
  - Rapidly building and disseminating institutional knowledge among members of a development team
  - The goal is to give all developer
- **Simplicity**
  - Extreme Programming encourag
  - "You ain't gonna need it"
- **Feedback**
  - Feedback from the system: by w
  - Feedback from the customer: Th
  - Feedback from the team: planni
- **Courage**
  - One is the commandment to alw
  - knowing when to throw code aw to create that source code
  - persistence: A programmer migh day, if only they are persistent.
- **Respect**
  - For others as well as self-respec
  - never commit changes that brea peers.
  - always striving for high quality a



# Content

1. Non-agile Software Processes
2. Agile Software Development
3. eXtreme Programming (XP)
4. Scrum

# Scrum

Scrum is an agile process that allows developers to focus on delivering the highest business value in the shortest time.

"tries to go the distance as a unit, passing the ball back and forth"

# Scrum

- Originally proposed by Schwaber and Beedle
- Scrum — distinguishing features
  - Development work is partitioned into "packets"
  - Testing and documentation are on-going as the product is constructed
  - Work occurs in "sprints" and is derived from a "backlog" of existing requirements
  - Meetings are very short and sometimes conducted without chairs
  - "demos" are delivered to the customer with the time-box allocated

# Roles in Scrum

## Scrum Roles: A different way of thinking, a better way to drive success

Scrum roles differ from traditional project roles.
By collaborating, a Scrum team delivers more business value, faster.

Presents the customer

### Product Owner
The Holder of Product Value

Determines what needs to be done and sets the priorities to deliver the highest value

Traditional approach:
Controls the work

### ScrumMaster®
The Servant Leader

Protecting the Scrum process and preventing distractions

Traditional approach:
No equivalent

NOT the team leader

### Development Team
The Self-Organizing Group

Takes on and determines how to deliver chunks of work in frequent increments

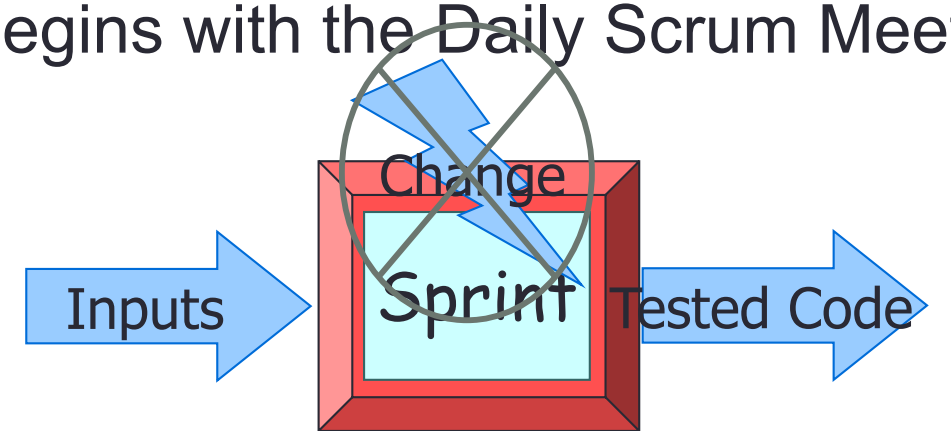
Traditional approach:
Gets told what to do by the project manager

about 7 members, self-organizing

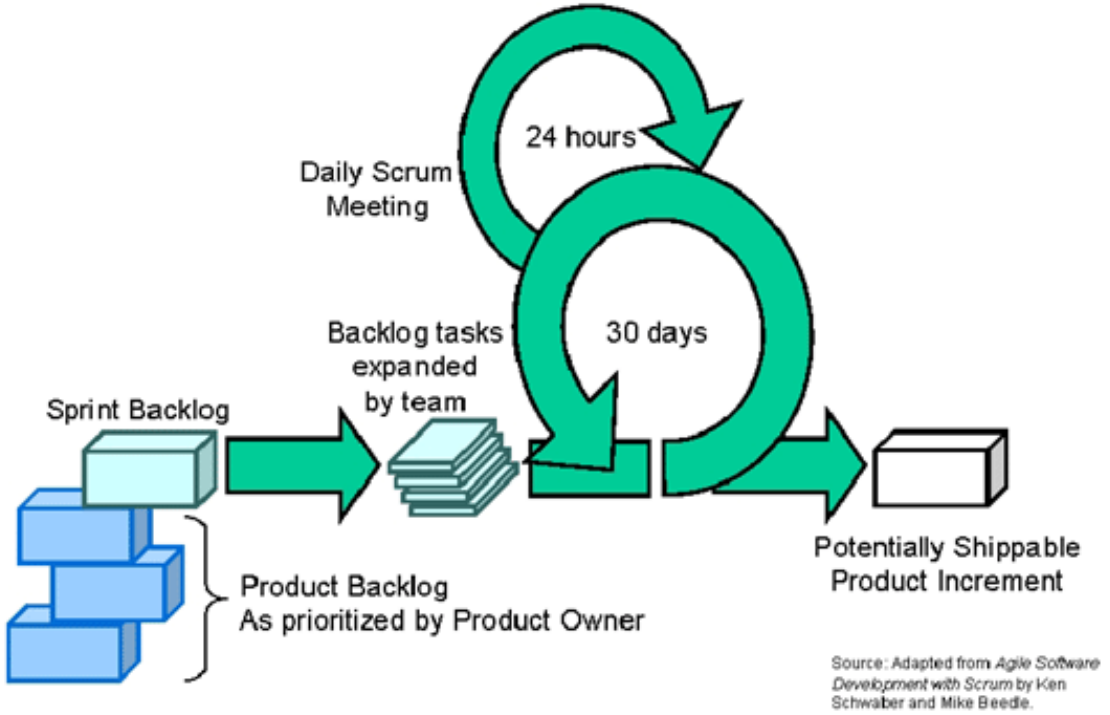
Scrum Alliance®

# Sprint

- Scrum projects make progress in a series of “sprints”
- Target duration is one month
  - +/- a week or two: a constant duration leads to a better rhythm
- Product is designed, coded, and tested during the sprint
- NO changes during the sprint
  - Plan sprint durations around how long you can commit to keeping change out of the sprint
- NO outside influence can interfere with the Scrum team during the Sprint
- Each Sprint begins with the Daily Scrum Meeting



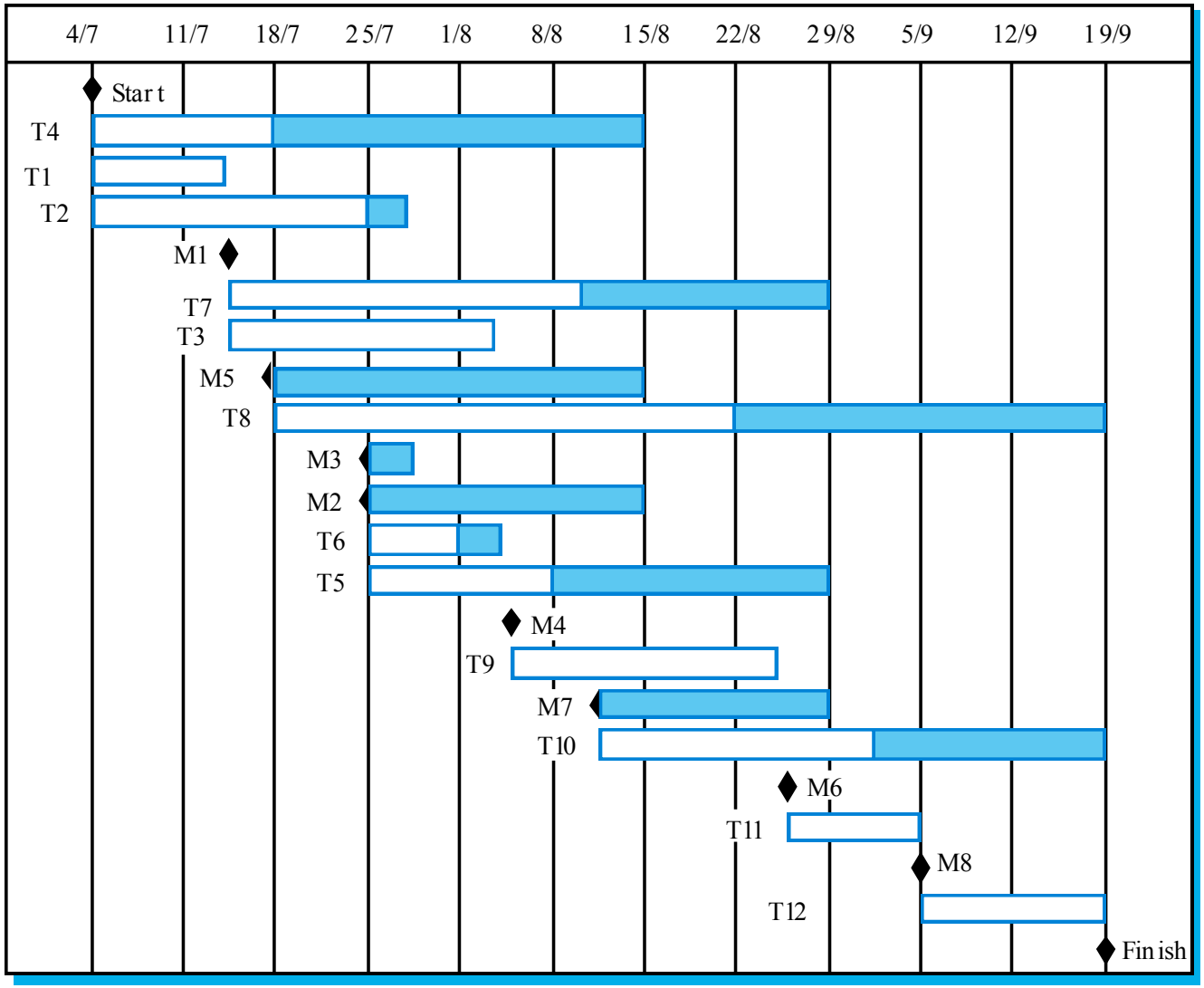
# Scrum Process




## Time-Boxed vs Activity-Based Planning

- **TB:** The project is decomposed into fixed-length time boxes, within which development activities are performed
- **AB:** The project is decomposed into a set of activities which their completion represent a Milestones of the project
- Scrum realized **time-boxed** development through **Sprints**, which are 2-4 weeks

# Scrum Process



# Scrum Practices

- **The Sprint Planning Meeting**
  - Is attended by the Product Owner, Scrum Master, the entire Scrum Team, also any interested and appropriate management or customer representatives

1. Product Owner describes highest priority features to the Team
   - The Product Owner selects the ideal backlog for the coming Sprint and communicates its meaning and importance to the team.
   - Team asks questions for clarification of PB items

2. Collectively, the Scrum team and the product owner define a **sprint goal**
   - A short description of what the sprint will attempt to achieve.
   - The success of the sprint will later be assessed during theSprint Review Meeting against the sprint goal, rather than against each specific item selected from the product backlog.

3. Team decides (separately) what the can commit to delivering in the Sprint
   - The Team decides how much it can commit to delivering in the coming Sprint.
   - The Product Owner answers questions but does not direct the team's choices.
   - The outcome is the Sprint Backlog

# Scrum Practices

- **The Sprint Review Meeting**
  - At the end of each sprint a sprint review meeting is held. (2 hours)
  - Team demonstrates product increment to product owner: A demo of the new features
  - Informality is encouraged. PowerPoint is discouraged.
  - A sprint review meeting **should not become a distraction or significant detour for the team**; rather, it should be a natural result of the sprint.
  - During the sprint review the project is assessed against the sprint goal determined during the Sprint planning meeting.
    - Ideally the team has completed each product backlog item brought into the sprint,
    - But it is more important that they achieve the overall goal of the sprint.

- **The Sprint Retrospective**
  - Time boxed to three hours, at the end of each Sprint
  - Team, Scrum Master, and (optionally) Product Owner review the last Sprint
    - **What went well?**
    - **What can be improved?**

# Scrum Practices

- **The Daily Scrum**
  - Time boxed to fifteen minutes! **EVERY DAY**
  - The Team and the Scrum Master (PO is also good to attend)
  - is not used as a problem-solving or issue resolution meeting.
    - Issues that are raised are taken offline and usually dealt with by the relevant sub-group immediately after the daily scrum
  - member provides answers to the following three questions
    - **What have you accomplished since yesterday?**
    - **What are you working on today?**
    - **Are there any impediments in your way?**
  - The daily scrum **is not a status update meeting** in which a boss is collecting information about who is behind schedule
    - Rather, it is a meeting in which team members make commitments to each other.
  - *Sample Impediments:*
    - My ____ broke and I need a new one today.
    - I still haven't got the software I ordered a month ago.
    - I need help debugging a problem with _____.
    - I'm struggling to learn _____ and would like to pair with someone on it.
    - I can't get the vendor's tech support group to call me back.
    - Our new contractor can't start because no one is here to sign her contract.
    - I can't get the ____ group to give me any time and I need to meet with them.
    - The department VP has asked me to work on something else "for a day or two."

# Scrum FAQs

- Why daily?
  - “How does a project get to be a year late?”
    - “One day at a time.”
      - Fred Brooks, The Mythical Man-Month.
- Can Scrum meetings be replaced by emailed status reports?
  - No
    - Entire team sees the whole picture every day
    - Create peer pressure to do what you say you’ll do

# Scrum Artifacts

- **The Product Backlog**
  - The Product Backlog is the ***master list of all functionalities*** desired in the product
  - It is **NOT** necessary to start a project with a lengthy, upfront effort to document all requirements.
  - Product backlog items can be technical tasks ("Refactor the Login class to throw an exception") or more user-centric (“Withdraw money from my bank account").
  - XP User Stories is a good approach to fill the Product Backlog
  - Product Owner writes and prioritize the product backlog
  - Scrum master can update it along the sprints

- Sample Product Backlog (*based on XP User Stories*)
  - As a user, I want to reserve a hotel room
  - As a user, I want to cancel a reservation
  - As a vacation planner, I want to see photos of the hotels
  - As a frequent flier, I want to rebook a past trip, so that I save time booking Trips

# Sample Product Backlog

| Backlog item | Estimate |
|---|---|
| Allow a guest to make a reservation | 3 |
| As a guest, I want to cancel a reservation. | 5 |
| As a guest, I want to change the dates of a reservation. | 3 |
| As a hotel employee, I can run RevPAR reports (revenue-per-available-room) | 8 |
| Improve exception handling | 8 |

# Sample 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 |
| | 19 | **Add icons for v1.1 or 2.0** | - | - |
| | - | **Pedigree Manager** | - | - |
| | 20 | Validate Derived kindred | 4 | KH |
| **Medium** | | | | |
| | - | **Explorer** | - | - |
| | 21 | Launch tab synchronization (only show queries/analyses for logged in users) | 8 | T&A |
| | 22 | Delete settings (?) | 4 | T&A |

Estimates have been developed by the developers but it is understood that they are very imprecise and are useful only for rough assignments of tasks into the various sprints.

- *Point Estimation*
- *Time Estimation*

# Scrum Artifacts

- **The Sprint Backlog**
  - The list of tasks that the Scrum team is committing that they will complete in the current sprint.
  - Items on the sprint backlog are drawn from the Product Backlog, and Detailed into smaller list of things needed to be done
  - Selected based on the priority of in the product backlog
  - *Due by next sprint !!*

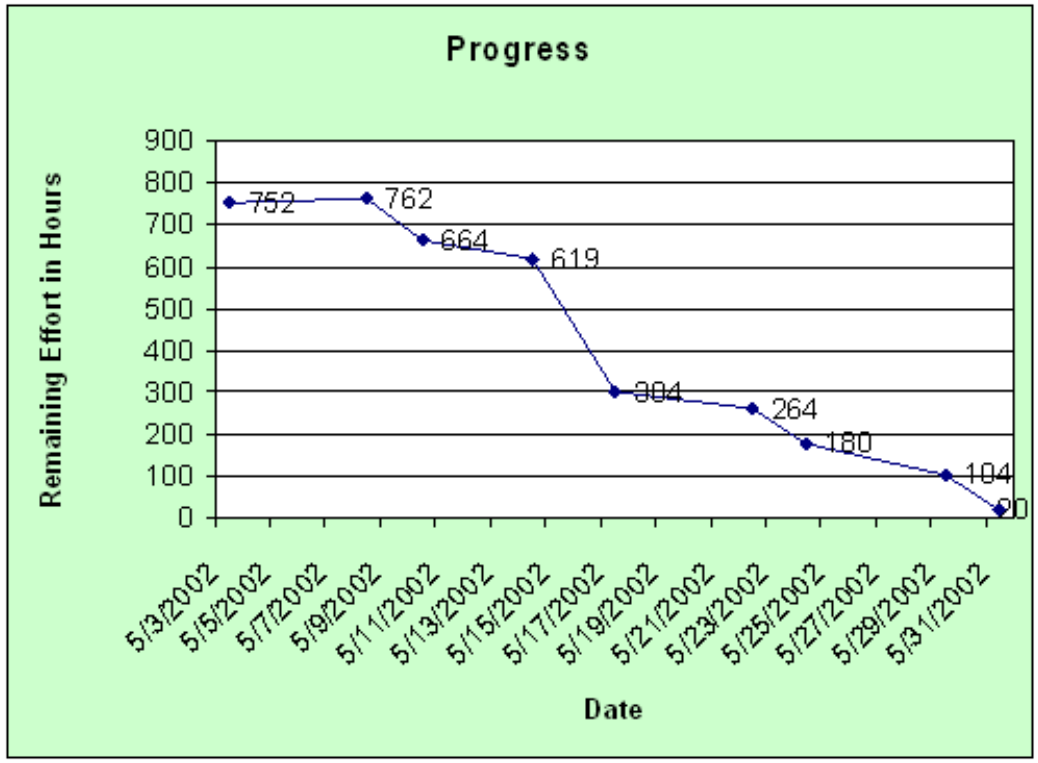
- Sample Sprint Backlog
  - As a user, I want to reserve a hotel room
    - Add hotel table to the database – 1 hr
    - Write Ajax code to display reservation – 4 hrs
    - Write code to enter reservation in the database – 4 hrs
  - As a user, I want to cancel a reservation
    - Display the user's current reservations – 4 hrs
    - Add a cancel button next to each reservation – 1 hr

# Sample Sprint Backlog

# Scrum Artifacts

- **The Sprint Burndown Chart**
  - Shows the project progress on a daily basis

During the Sprint the ScrumMaster maintains the sprint backlog by updating it to reflect which tasks are completed and how long the team thinks it will take to complete those that are not yet done. The estimated work remaining in the sprint is calculated daily and graphed, resulting in a sprint burndown chart like this one:

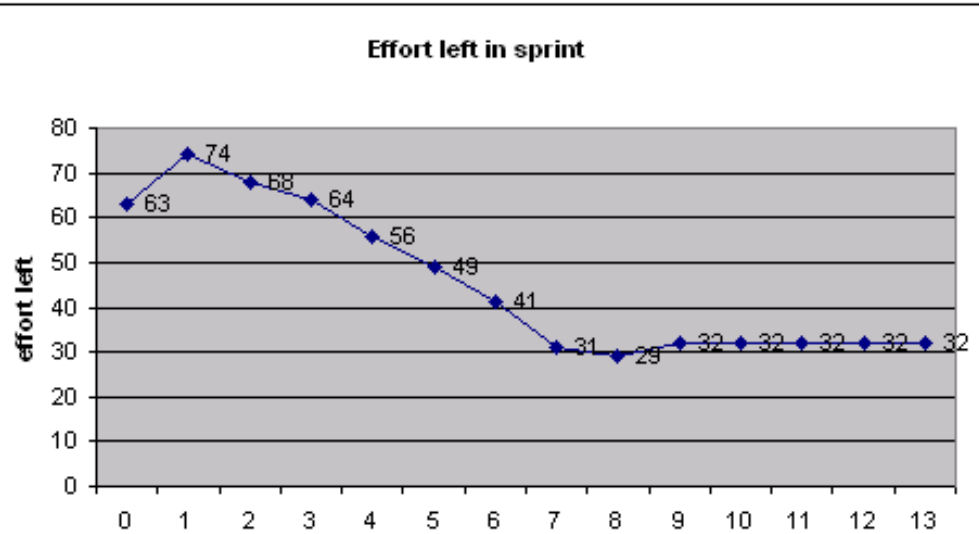


# Sample Sprint Backlog / S. Burndown Chart

**Sprint 3. Plug in the Real Weather**

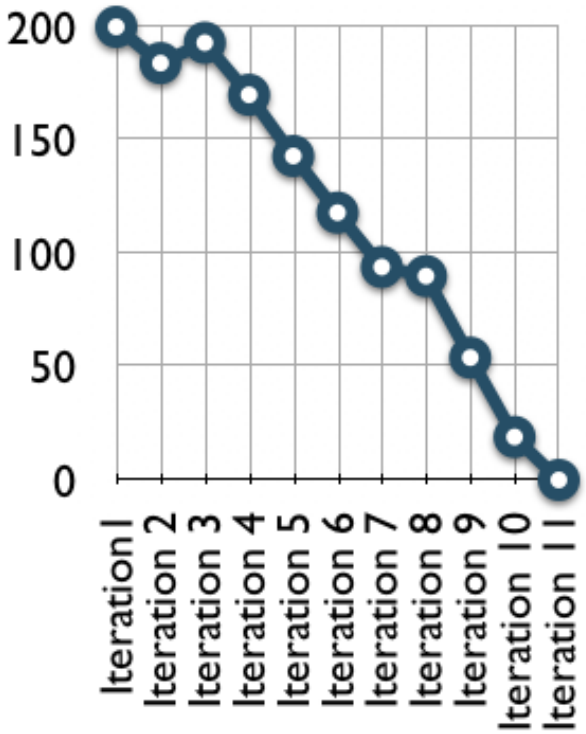
| Story ID | Story/task | days in sprint / effort left | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| | | 63 | 74 | 68 | 64 | 56 | 49 | 41 | 31 | 29 | 32 | 32 | 32 | 32 | 32 |
| 10 | **Fetch one day temperature data from the weather provider system** | | | | | | | | | | | | | | |
| | Make our server connect and authenticate to the provider system | 4 | | | | | | | | | | 3 | 3 | 3 | 3 |
| | Read provider's data directory | 8 | 7 | | | | | | | | | 0 | 0 | 0 | 0 |
| | Parse the current temperature out of the data | 6 | 6 | | | | | | | | | 1 | 1 | 1 | 1 |
| | Push the temperature data to the client | 16 | 16 | | | | | | | | | 0 | 0 | 0 | 0 |
| 11 | **Fetch rain, snow, etc details from the provider** | | | | | | | | | | | | | | |
| | Parse snow/rain data from the provider's data | 4 | 4 | | | | | | | | | 0 | 0 | 0 | 0 |
| | Push the snow/rain data to the client | 4 | 4 | | | | | | | | | 0 | 0 | 0 | 0 |
| | Redesign client screen a bit | | | | | | | | | | | 3 | 3 | 3 | 3 |
| | Refactor the server code | | | | | | | | | | 4 | 4 | 4 | 4 | 4 |
| 12 | **Fetch several days data from the provider** | | | | | | | | | | | | | | |
| | Parse the weather data in day packs | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| | Push several days data to the client | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 |
| 13 | **Auto-refresh feature** | | | | | | | | | | | | | | |
| | Make the client ping server once per 4 hours | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| | Make the server update the client | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |

The team underestimated the difficulties of connecting to the provider's CORBA server. Even though some work has been already done, the remaining amount is much bigger, than the original estimation for the whole task



# Scrum Artifacts

- Release Burndown Chart
  - The team tracks its progress against a release plan by updating a release burndown chart at the end of each sprint.
  - The horizontal axis of the release burndown chart shows the sprints;
  - The vertical axis shows the amount of work remaining at the start of each sprint.



# Sample Product Backlog / R. Burndown Chart

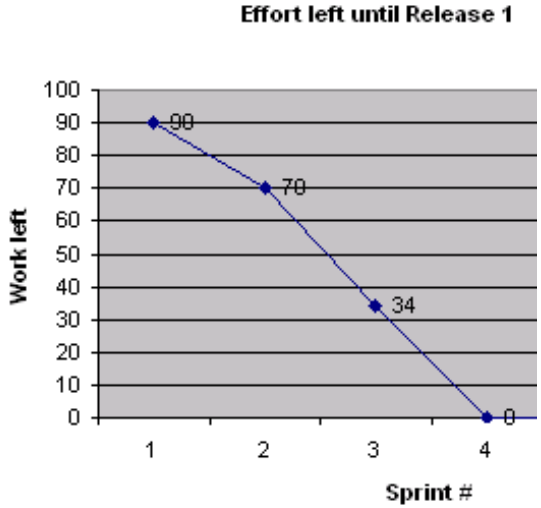
## Weather on Mobile

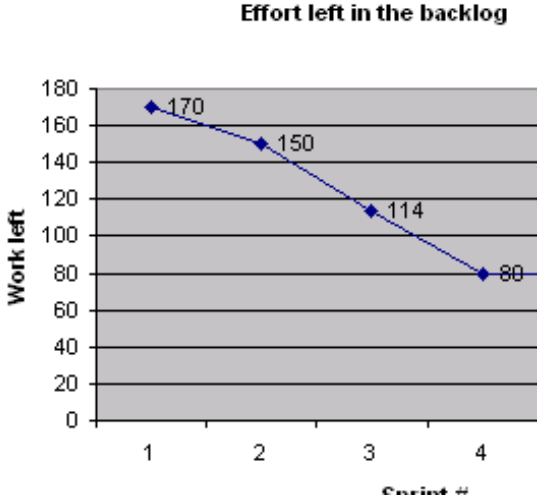
http://agilesoftwaredevelopment.com/scrum/simple-product-backlog

| ID | Description | Sprint # | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|---|
| | **Effort needed for Release 1 as in the beginning of the sprint** | | 90 | 70 | 34 | 0 | 0 | 0 |
| 1 | Set up continuous integration system | | 5 | 0 | 0 | 0 | 0 | 0 |
| 2 | Create compilable application skeleton | | 5 | 0 | 0 | 0 | 0 | 0 |
| 3 | Display current temperature in a simplest possible way | | 13 | 0 | 0 | 0 | 0 | 0 |
| 4 | Set up the web server for serving weather data | | 3 | 0 | 0 | 0 | 0 | 0 |
| 5 | Implement stubby WeatherML support on the server side | | 13 | 0 | 0 | 0 | 0 | 0 |
| Sprint 1 | *Make sample data go from server to device* | | | | | | | |
| 6 | Graphics support on the client side | | 20 | 0 | 0 | 0 | 0 | 0 |
| 16 | Make the graphics library draw some icon and sample temperature text | | - | 13 | 0 | 0 | 0 | 0 |
| 17 | Draw the real weather screen | | - | 8 | 0 | 0 | 0 | 0 |
| 7 | Implement support for several days | | 8 | 8 | 0 | 0 | 0 | 0 |
| 8 | Implement support for rain, snow, etc. icons | | 2 | 2 | 0 | 0 | 0 | 0 |
| 9 | City changing support | | - | 5 | 0 | 0 | 0 | 0 |
| Sprint 2 | *Minimal working version* | | | | | | | |
| 10 | Fetch one day temperature data from the weather provider system | | ? | | | | | |
| 11 | Fetch rain, snow, etc details from the provider | | | | | | | |
| 12 | Fetch several days data from the provider | | | | | | | |
| 13 | Auto-refresh feature | | | | | | | |
| Sprint 3 | *Plug in the real weather data* | | | | | | | |
| **Release 1** | **Sellable version** | | | | | | | |
| 14 | Inject simulated ads from the test server | | | | | | | |
| 15 | Plug real ads in | | | | | | | |
| 18 | Change current city automatically according to the cell info | | 40 | 40 | 40 | 40 | 40 | 40 |
| Sprint 4 | *Advertisements support* | | | | | | | |
| **Release 2** | **Ad-supported version** | | | | | | | |
| | **Effort in the whole backlog** | | 170 | 150 | 114 | 80 | 80 | 80 |

In a real product backlog, items are more likely to be in the form of user stories. E.g. this item could be stated as "As a user I want to see the real weather data so that I could know what to wear before leaving home"

Backlog state taken after the end of sprint 3 = after release 1



# Adjusting the Sprint Backlog

- The team does its best to pull the right amount of work into the sprint
- But sometimes too much or too little work is pulled in during the Sprint planning meeting.
- In this case the team needs to add or remove tasks.


In the above sprint burndown chart you can see that the team had pulled in too much work initially and still had nearly 600 hours to go on 5/16/02. In this case the Product Owner was consulted and it was agreed to remove some work from the sprint, which resulted in the big drop on the chart between 5/16/02 (619 hours) and 5/17/02. From there the team made good consistent progress and finished the sprint successfully.

# Agile vs Waterfall

| Waterfall | Agile |
|---|---|
| Predictive | Adaptive |
| Low uncertainty and low urgency | High uncertainty or high urgency |
| Not new product development | New product development |
| Traditional, process-oriented staff | Right sort of people on staff (innovative culture) |
| Low level of direct customer participation | High level of direct customer participation |
| More time and money | Less time and money |

# Agile vs Spiral

| Spiral | Agile |
|---|---|
| Repetitive | Non-repetitive |
| Takes longer time | Takes less time |
| Iteration through all steps | Iteration through certain steps |
| More time and money | Less time and money |
| More solid results | Less solid results |

# References


- An Introduction to Agile Software Development: http://www.agilealliance.org
- http://agilemanifesto.org/
- http://www.mountaingoatsoftware.com/scrum/overview