Software Development Project

Morgan Ericsson
<morgan.ericsson@chalmers.se>

Hello

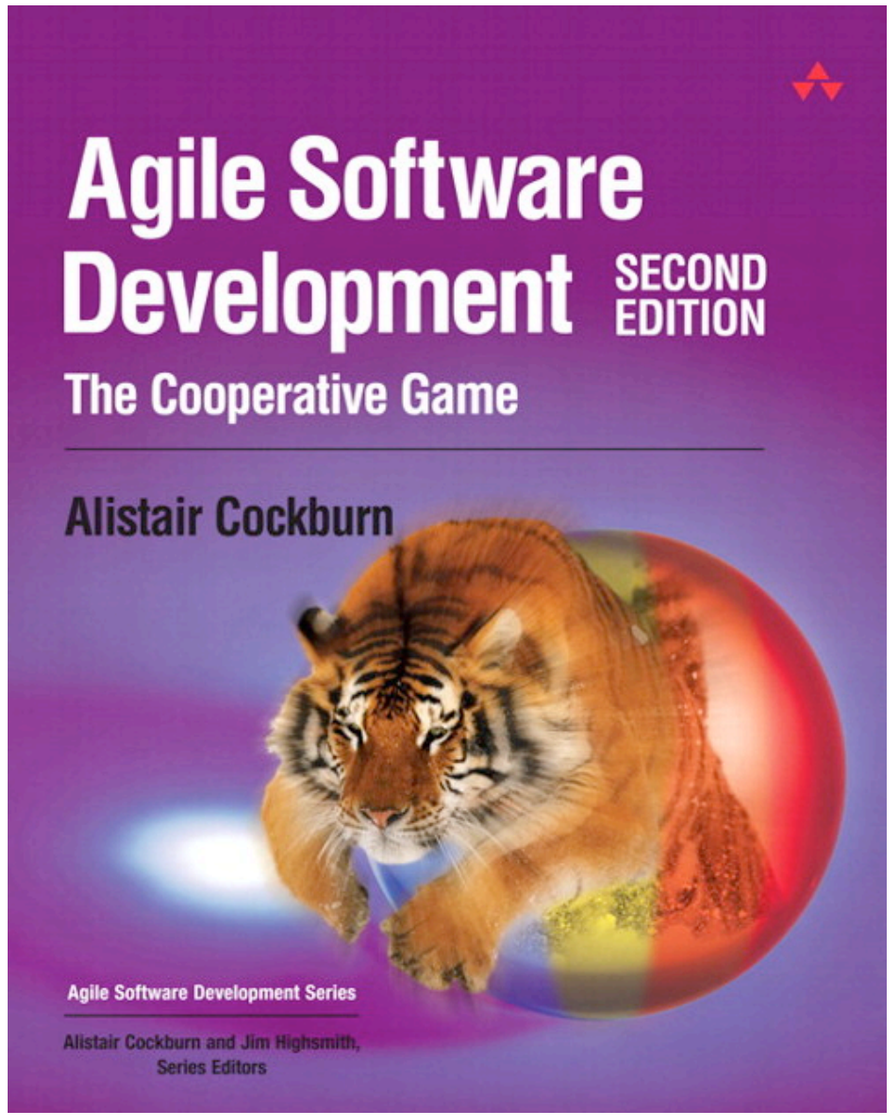
- Morgan Ericsson
  - morgan.ericsson@(gu|chalmers).se
    - @moercth
- Lectures and administration
- Less than 3 weeks at GU/Chalmers, so be gentle!

Staff

- TA:s
- Emil Alégroth (emil.alegroth@chalmers.se)

Textbook

- Cockburn, A., (2009) Agile Software Development, 2ed
- Papers
- Online resources and lecture material



Practical Details

- https://github.com/morganericsson/EDA397
  - course material (vc'd)
  - wiki
    - issue tracking
- @moertech (with #EDA397)
  - updates
- Further resources may be added during the course...

Practical Details (cont'd)

- Schedule
  - 0-3 lectures per week
  - 0-2 workshops per week
- So, 3 scheduled activities per week
  - even if there is no lecture, we will be available and you can (should!) use the rooms/time to work

Examination

- Project (teams)
  - final product
  - artifacts
- Report (individual)
  - post-mortem experience report
- Written exam

Project

- Develop an Android app for a customer
- Work in predetermined teams
- You will be assigned teams and meet with the customer during course week 2
  - and get all the details!

Environment

- We strive to create a realistic scenario/ environment
- We rely on a number of real-world services and tools, e.g.
  - Android (SDK)
  - GitHub / BitBucket
  - ..

Outcomes

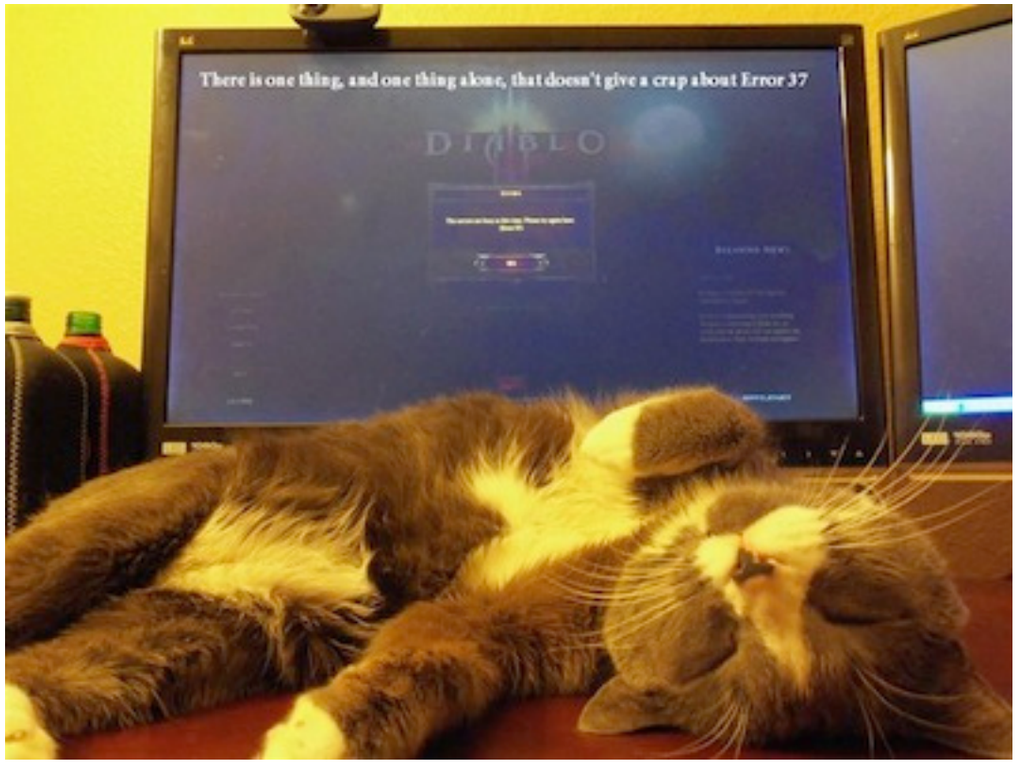
- You will learn a lot, e.g.
  - the software development process
  - useful tools and APIs
- By doing (a lot) and failing (a lot)
- And hopefully have fun while doing it!

Week 1

- Intro to course and development process
- Intro to Android and tools/services
- You should:
  - complete the survey
  - get everything working on your computer
  - play around with the Android SDK and Git
  - read the recommended reading

Week 1

- Tuesday: Course and Agile intro
- Thursday: Intro to Android
- Wednesday: Getting started (tutorial)
- Friday: Getting started (tutorial)
- Schedule sessions with TAs.



The Making of a Fly: The Genetics of Animal Design (Paperback)
by Peter A. Lawrence

‹ Return to product information

Always pay through Amazon.com's Shopping Cart or 1-Click.
Learn more about Safe Online Shopping and our safe buying guarantee.

Price at a Glance

List Price:	~~$70.00~~
Used:	from $35.54
New:	from $1,730,045.91

Have one to sell? Sell yours here

All | New (2 from $1,730,045.91) | Used (15 from $35.54)

Show ⦿ New ○ Prime offers only (0) — Sorted by Price + Shipping

New 1-2 of 2 offers

Price + Shipping	Condition	Seller Information	Buying Options
$1,730,045.91 + $3.99 shipping	**New**	Seller: **profnath** Seller Rating: ★★★★★ **93% positive** over the past 12 months. (8,193 total ratings) In Stock. Ships from NJ, United States. Domestic shipping rates and return policy. Brand new, Perfect condition, Satisfaction Guaranteed.	Add to Cart or Sign in to turn on 1-Click ordering.
$2,198,177.95 + $3.99 shipping	**New**	Seller: **bordeebook** Seller Rating: ★★★★★ **93% positive** over the past 12 months. (125,891 total ratings) In Stock. Ships from United States. Domestic shipping rates and return policy. New item in excellent condition. Not used. May be a publisher overstock or have slight shelf wear. Satisfaction guaranteed!	Add to Cart or Sign in to turn on 1-Click ordering.

Software Development is Difficult/Complex!

- The problems of characterizing the behavior of discrete systems.
- The flexibility possible through software
- The complexity of the problem domain
- The difficulty of managing the development process

"The complexity of software is an essential property, not an accidental one"

1. What should I do?

"Binary search is an elegant but simple algorithm that many of you have seen. The basic idea is to start with two inputs: a sorted array and a key to search for. If the key is found in the array, the index of the key is returned. Otherwise, an indication that the search failed is returned. What binary search does is to look first at the element in the middle of the array: if it is equal to the key, return the index; if it is less than the key, perform binary search on the "top half" of the array (not including the middle element); and if it is greater than the key, perform binary search on the "bottom half" of the array (not including the middle element). Correct implementations of the algorithm run in $O(lg_2N)$, which means that the worst case for running the program will take time proportional to the (base 2) logarithm of N, where N is the length of the sorted array."

Open questions (some):

- How does binary search indicate that it did not find the key?
- Which "middle element" should be picked if the (sub)array's length is even (like the second step above)?
- What if a value appears multiple times in the sorted array and that value is matched by a key for a search? Which index gets returned?

2. Doing it!

```
public static int search(int key, int[] a, int first, int last)
{
        if (last <= first)
            return -1;

        int mid = (first + last)/2;
        if (key < a[mid])
            return search(key, a, first, mid - 1);
        if (key > a[mid])
            return search(key, a, mid + 1, last);

        return mid;
}
```

(Can you spot the bugs?)

3. Did I actually do it?

Build it and try a few values that should work...

```
Using array [ 0  1  2  3  4 ].
Found 2 at index 2
Found 0 at index 0
Found 3 at index 3
```

(Seems to work, but...)

What Did We Learn?

- A simple assignment can raise a number of questions, some without good answers ...
- A simple implementation can contain several bugs/issues/problems ...
- And the above may not be detected when evaluating
- How does this scale with the problem?

Software Crisis and Engineering

- Established as a reaction to the "Software Crisis"
  - software was inefficient
  - software did not meet requirements
  - projects ran over time/budget
  - projects were unmanageable and software unmaintainable

"The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem."

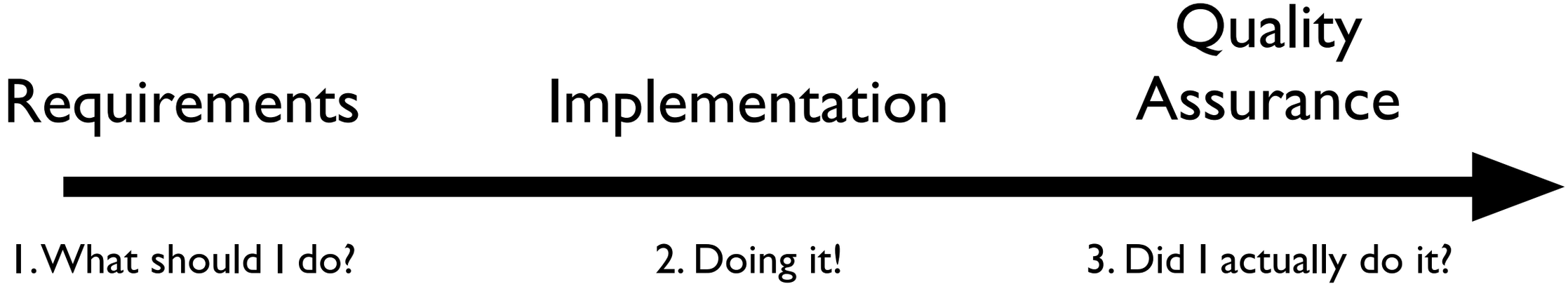
Software Engineering

- The branch of computer science that creates practical, cost-effective solutions to computing and information processing problems
- *"Application of engineering to software"*
  - *systematic*, *disciplined*, *quantifiable* approach to the *development*, *operation*, and *maintenance* of software
- Assure the quality of the process and the product

“Engineering Seeks Quality”

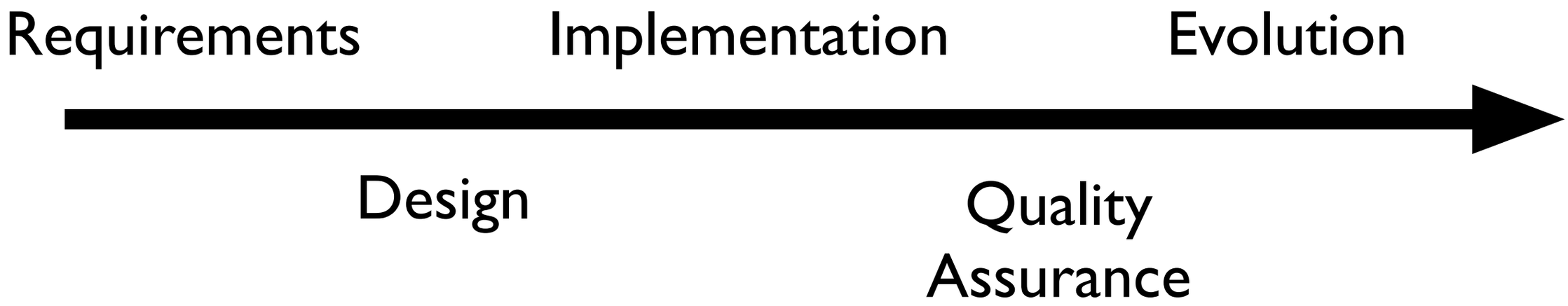
- So, the goal of software engineering is the production of quality software
- However, software is inherently complex
  - the complexity of software systems often exceeds the human intellectual capacity.
- The task of the software development team is to engineer the illusion of simplicity.

Three Steps Revisited

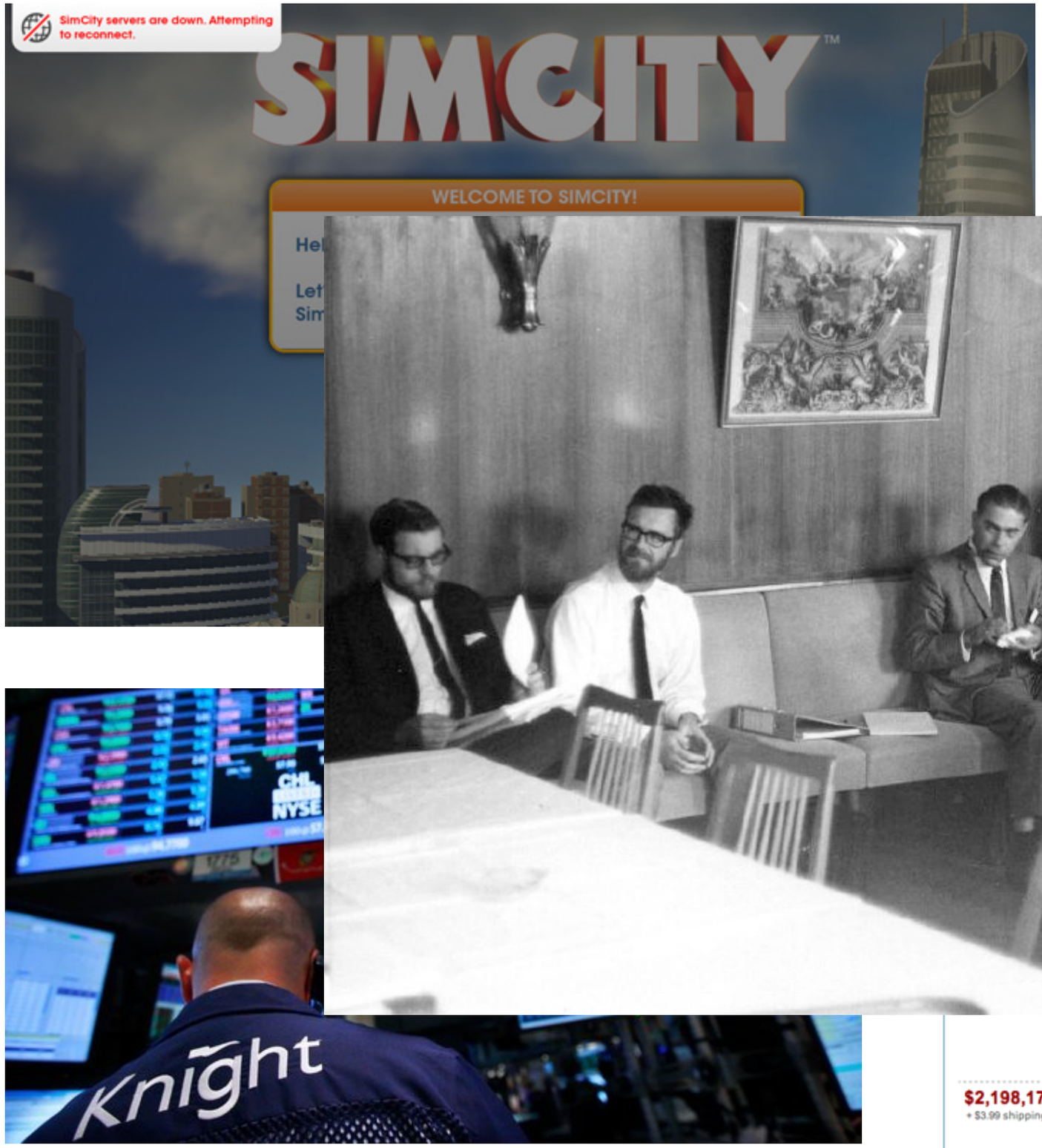


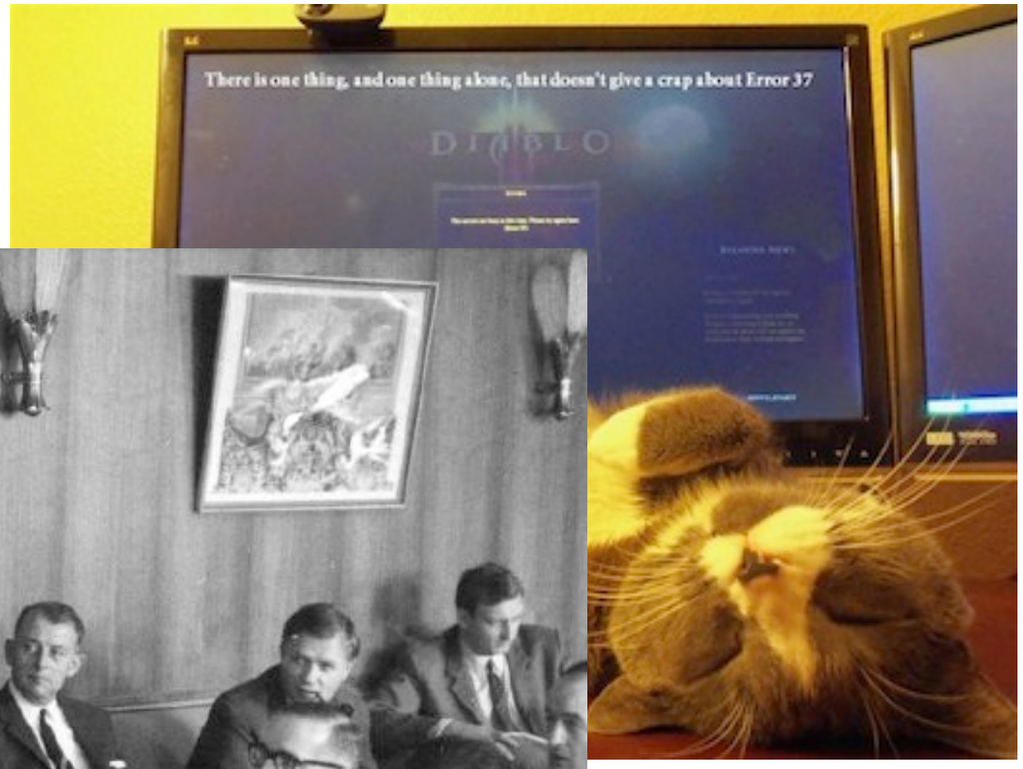
A sequential model of software development

Five Steps



A sequential model of software development





perback)

Price at a Glance

List Price: ~~$70.00~~
Used: from **$35.54**
New: from **$1,730,045.91**
Have one to sell? Sell yours here

Sorted by Price + Shipping

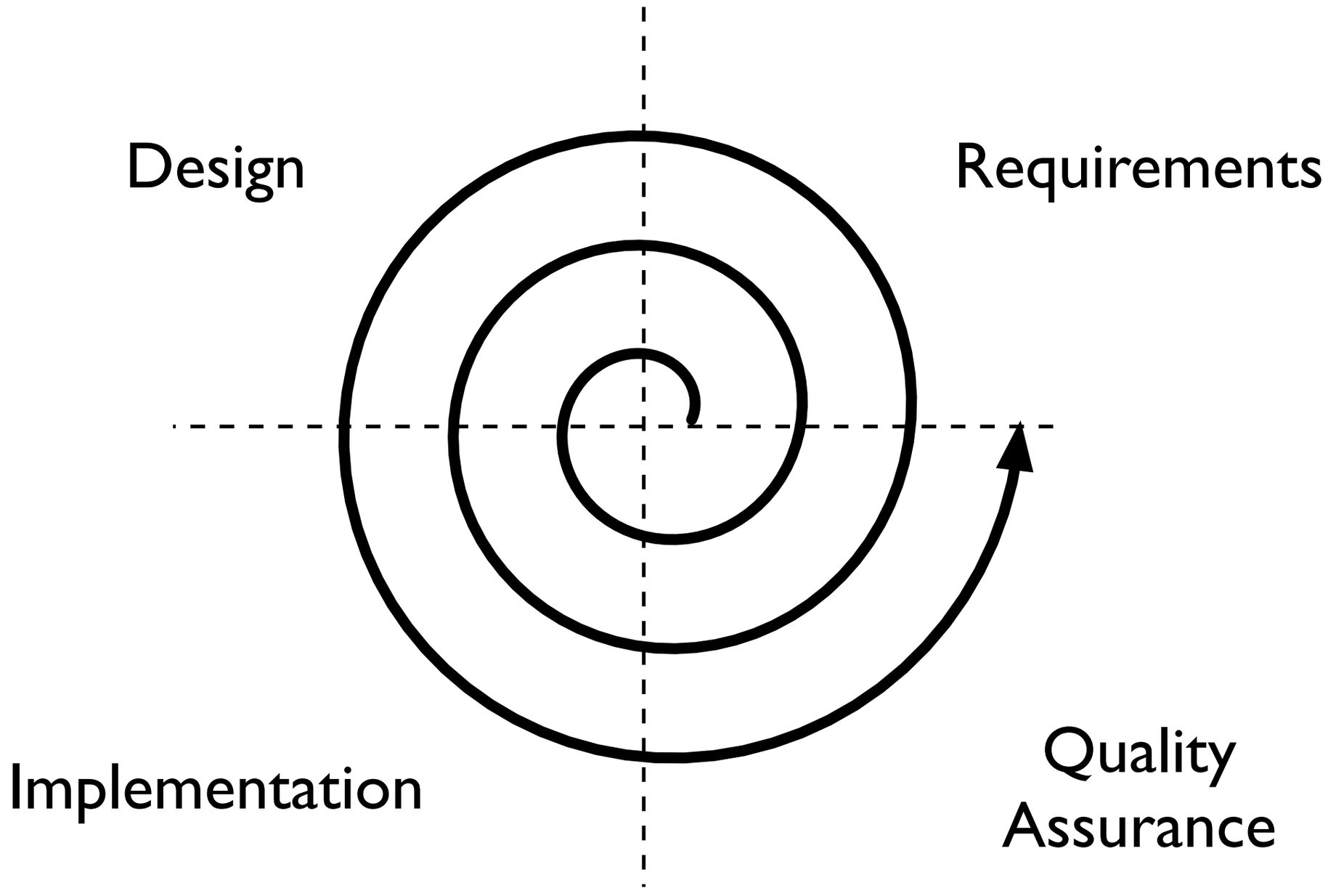
			Buying Options
		ver the past 12 months.	Add to Cart or Sign in to turn on 1-Click ordering.
		In Stock. Ships from NJ, United States. Domestic shipping rates and return policy. Brand new, Perfect condition, Satisfaction Guaranteed.	
$2,198,177.95 + $3.99 shipping	**New**	Seller: **bordeebook** Seller Rating: ★★★★★ **93% positive** over the past 12 months. (125,891 total ratings) In Stock. Ships from United States. Domestic shipping rates and return policy. New item in excellent condition. Not used. May be a publisher overstock or have slight shelf wear. Satisfaction guaranteed!	Add to Cart or Sign in to turn on 1-Click ordering.

Change is Ubiquitous

- Manage
  - change
  - uncertainty
  - quality

Production vs. Creation

Change is Ubiquitous



An iterative model of software development

Defined Process vs. Empirical Process

- Laying out a process that repeatedly will produce acceptable quality output is called defined process control
- When defined process control cannot be achieved because of the complexity of the intermediate activities, something called empirical process control has to be employed

Defined Process control

- planning heavy
- assumes (more) static environment
- longer iterations
- Change Management intensive
- typical pre-study heavy
- assumes good estimations
- control over Actual work (seen seen as bureaucratic)

Empirical Process control

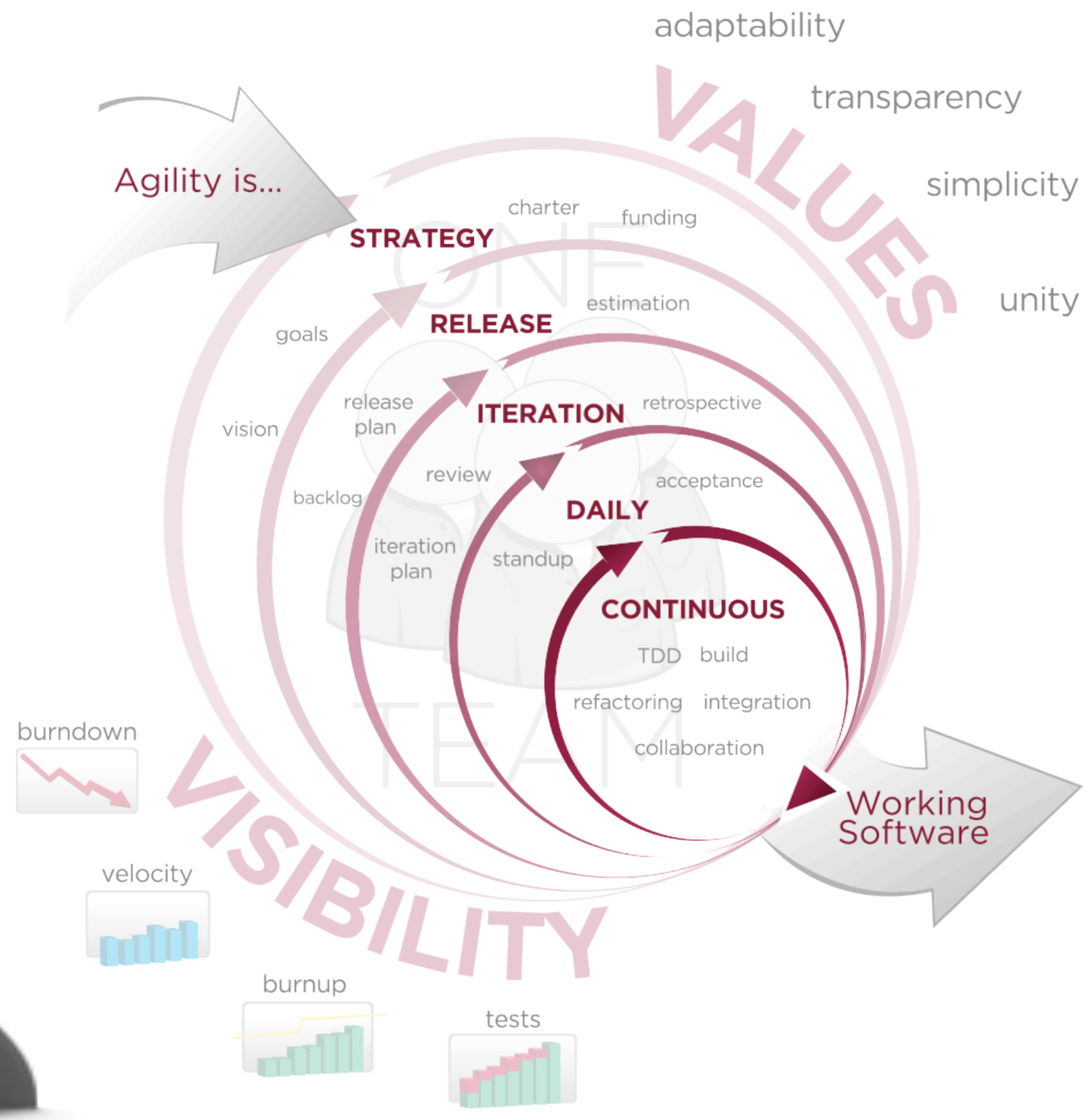
- change is reality
- shorter iterations
- problem vs. solution space (empowering the developers)
- just-enough (management, documentation, etc.)
- self organizing teams
- continuous “customer” interaction
- **NOT UNPLANNED, rather adaptive!!!**

Production vs. Creation

Going Agile

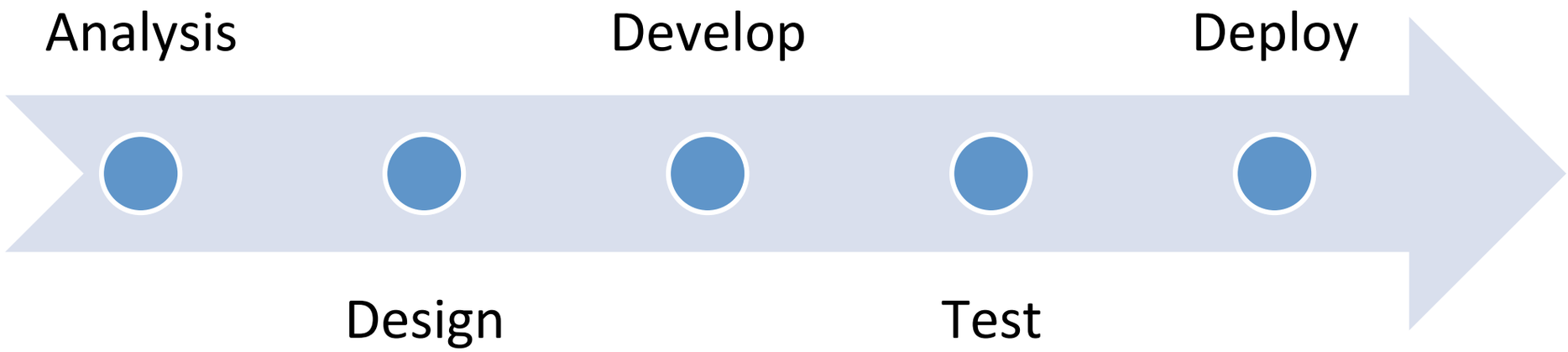
- Reaction against heavy-weight models mid 1990s
  - regulated, micro-managed, etc.
- Resulted in the development of light-weight models with focus on
  - in principle, a return to “earlier” methods

AGILE DEVELOPMENT

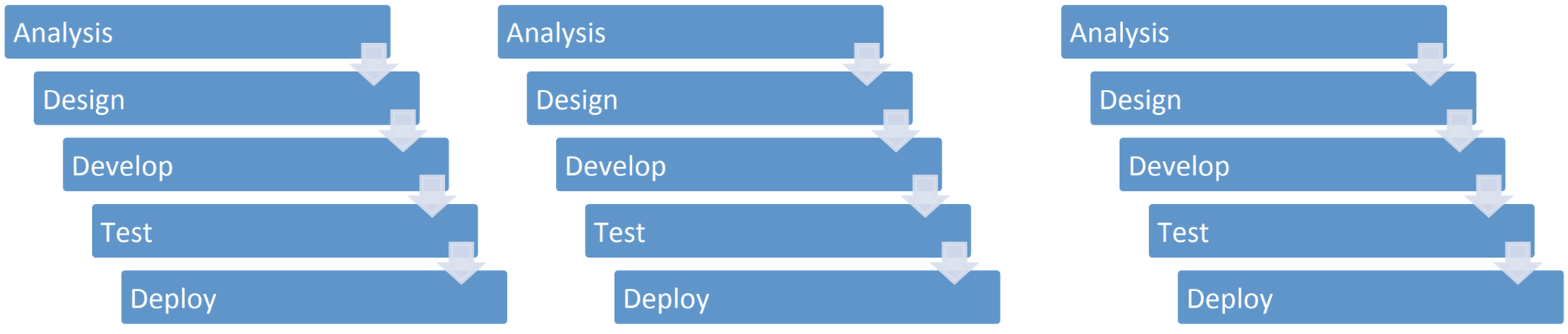


ACCELERATE DELIVERY

Non-agile



Agile



Agile Manifesto

- **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."

Principles behind the Agile Manifesto

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Principles behind the Agile Manifesto

- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Principles behind the Agile Manifesto

- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.

Principles behind the Agile Manifesto

- Simplicity – the art of maximizing the amount of work not done – is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

*"We were **doing incremental development as early as 1957**, in Los Angeles, under the direction of Bernie Dimsdale. He was a colleague of John von Neumann, so perhaps he learned it there, or **assumed it as totally natural**.*

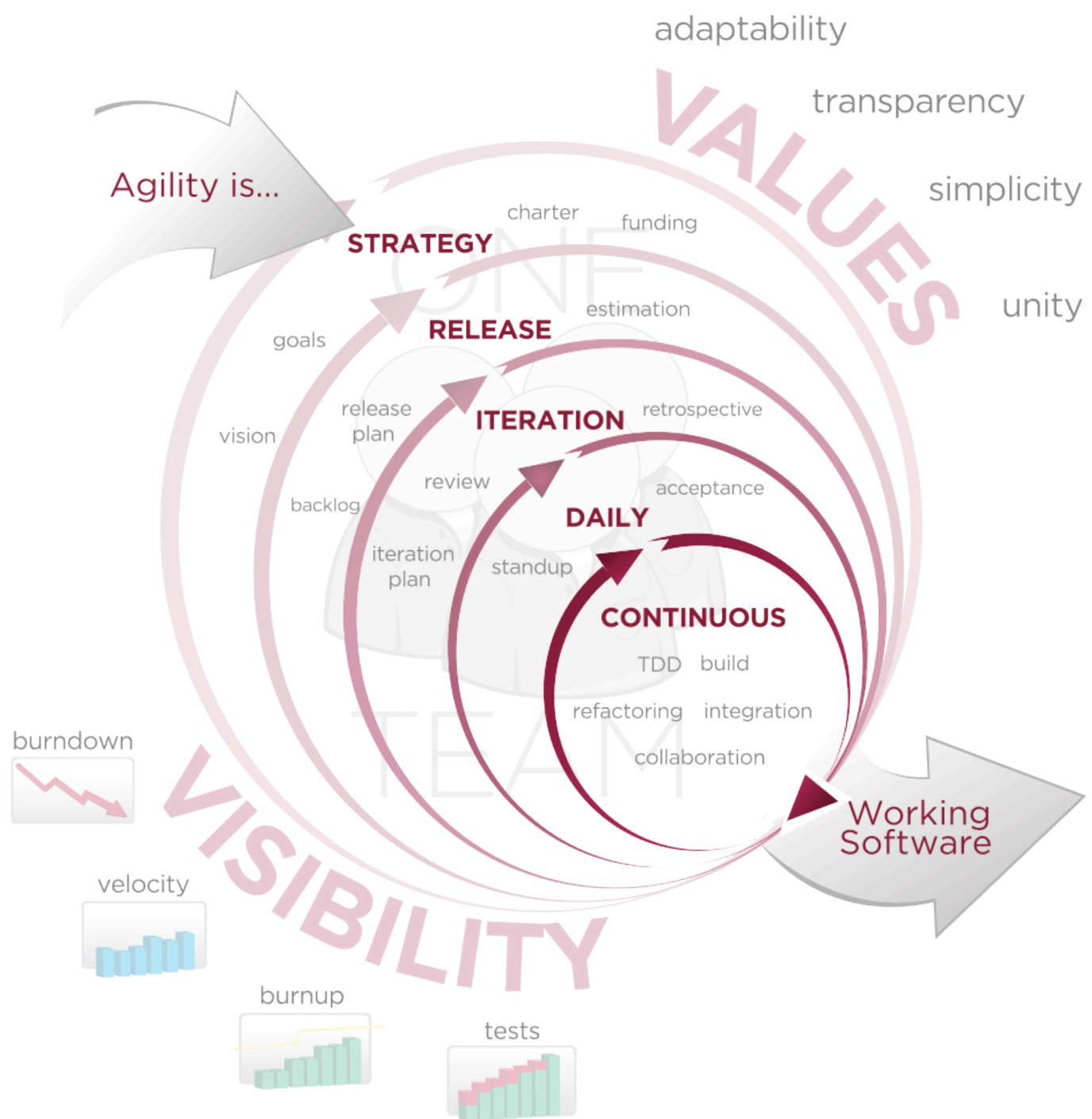
*I do remember Herb Jacobs developing a large simulation for Motorola, where the technique used was, as far as I can tell .... All of us, as far as I can remember, **thought waterfalling of a huge project was rather stupid, or at least ignorant of the realities**. I think what the waterfall description did for us was make us realize that we were doing something else, something unnamed except for 'software development.'"*

Gerald M. Weinberg

"What is new about Agile Methods is ***not the practices*** *they use, but their recognition of* ***people as the primary drivers*** *of project success, coupled with an intense* ***focus on effectiveness and maneuverability***"

Cockburn and Highsmith

AGILE DEVELOPMENT



ACCELERATE DELIVERY

Methodologies & Practices

- Scrum
- eXtreme Programming (XP)
- Kanban

Why Scrum?

- We need to do a better job of change management. We had too many outside distractions.
- We need customer feedback during the iterative development approach we're taking.
- The users gave us a huge list of requirements. We knew we weren't going to be able to deliver everything they wanted.
- Development took place in focused chaos, and there was no one to go to with questions. We need a way to structure the chaos somehow, because all NBOs must deal with that.
- We have been used to thinking in terms of years for development; now we have to turn out products in months.
- We're chasing an emerging market. It changes weekly.
- We wasted a lot of time estimating and developing test plans for features that we never developed.
- We should have cancelled this project earlier. It took almost two years to recognize that.
- We need well-defined phases and someone who is close enough to see progress and determine what we can check off.

Resing and Janoff (2000)

Roles

Commitment vs. Involvement

The Pigs

- Product owner
  - Represents interests of everyone with a stake in the project and resulting system
  - Responsible for initial/ongoing funding, initoal overall requirements, ROI objectives, release plans

The Pigs

- Scrum master
  - responsible for the Scrum process
  - teaching Scrum to everyone involved in the project
  - implements Scrum so that it fits within an organization's culture and still delivers the expected benefits for ensuring that everyone follows Scrum rules and practices.

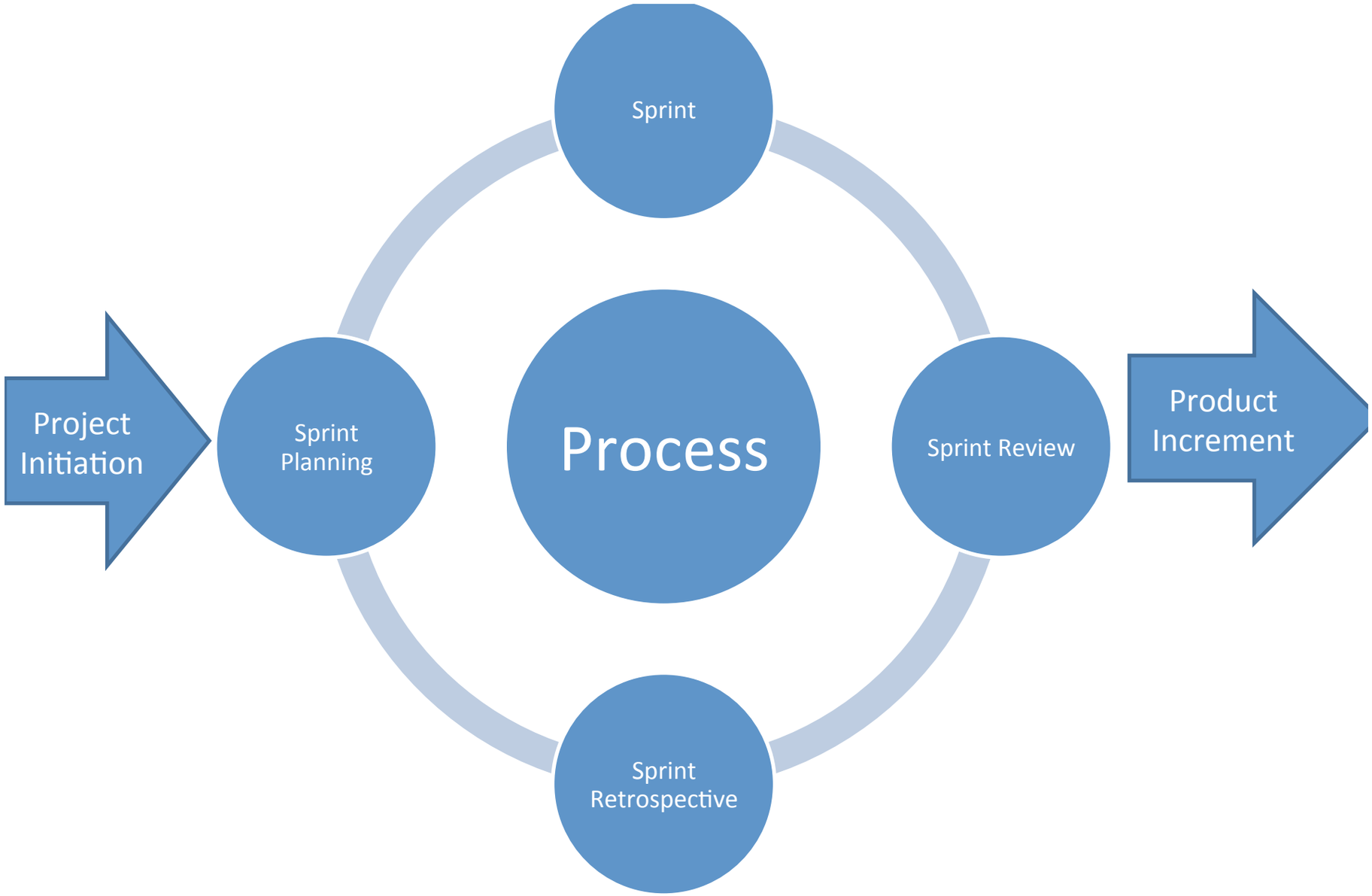
The Pigs

- Scrum team
  - Self-managing, self-organizing, and cross-functional
  - responsible for figuring out how to turn a list of requirements into an increment of functionality

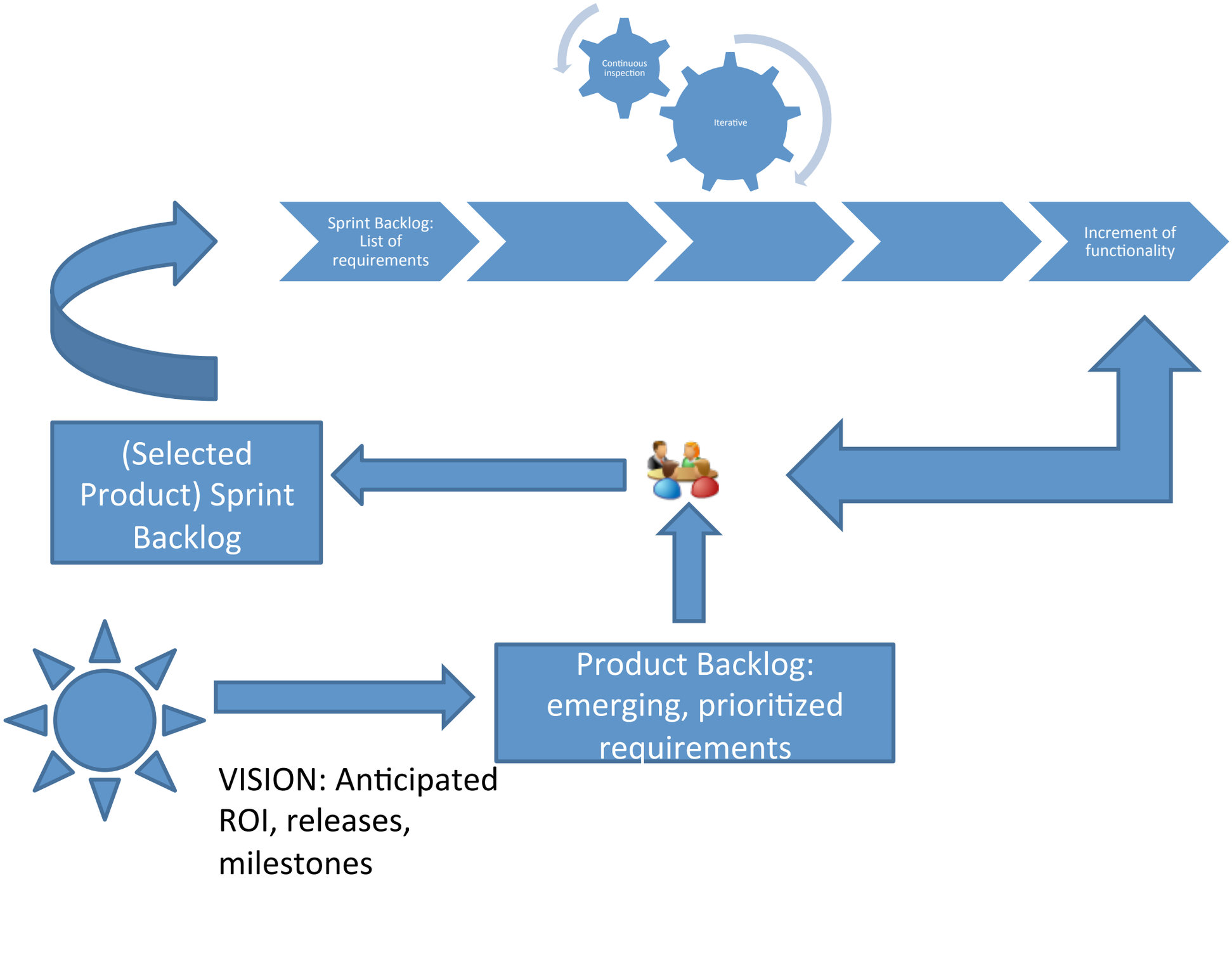
The Chickens

- Stakeholders
- Users

Scrum Process



Scrum Process



Product Vision

- What are the aims and objectives of the planned product
- Which markets to cover,
- Which competitors to compete,
- What is product's differentiation etc

Product Backlog

- The requirements for the system or product being developed by the project(s) are listed in the Product Backlog
- Product Owner is resonsible for the contents, prioritization, and availability of the Product Backlog

Product Backlog

- Never complete
- Merely an initial estimate of the requirements
- Evolves as the product and the environment in which it will be used evolves
- Dynamic - management constantly changes it to identify what the product needs to be appropriate, competitive, and useful.
- Exists as long as a product exists

Sprint Backlog

- Defines the work, or tasks, that a Team defines for turning the Product Backlog it selected for that Sprint into an increment of potentially shippable product functionality
- The Team compiles an initial list of these tasks in the second part of the Sprint planning meeting

Sprint Backlog

- Should contain tasks such that each task takes roughly 4 to 16 hours to finish
- Tasks longer than 4 to 16 hours are considered mere placeholders for tasks that have not yet been appropriately defined
- Only the Team can change it
- Highly visible, real-time picture of the current Sprint

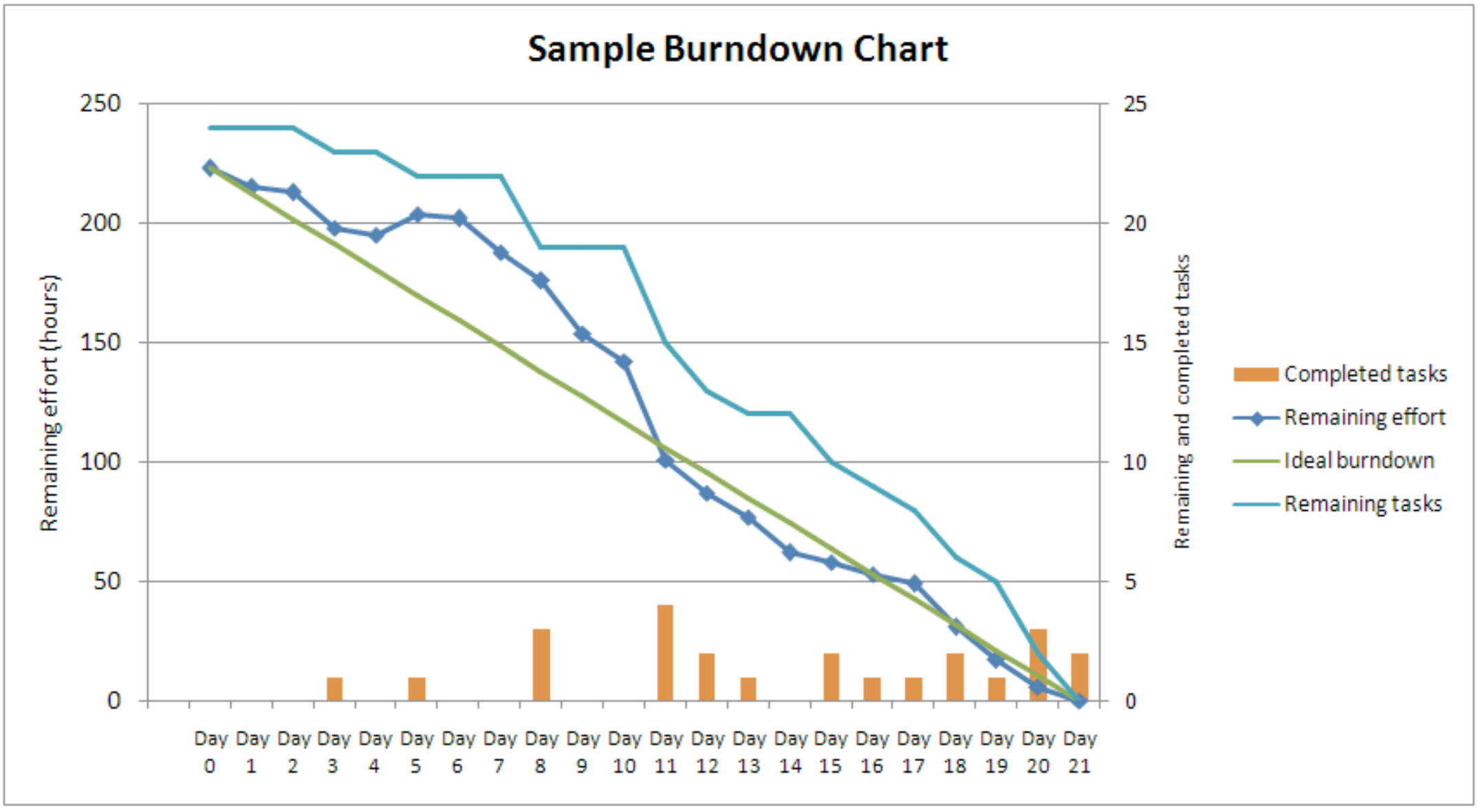
Burndown Chart

- Shows the amount of work remaining across time
- The Team compiles an initial list of these tasks in the second part of the Sprint planning meeting

Burndown Chart

- Excellent way of visualizing the correlation between the amount of work remaining at any point in time and the progress of the project Team(s) in reducing this work
- Allows to "what if" the project by adding and removing functionality from the release to get a more acceptable date or extend the date to include more functionality
- Highly visible, real-time picture of the current Sprint

Burndown Chart



Project Initiation

- Product Vision
  - might be vague at first, perhaps stated in market terms rather than system terms, but becomes clearer as the project moves forward
- Product Backlog
  - list of functional and nonfunctional requirements that, when turned into functionality, will deliver this vision
  - Prioritized so that the items most likely to generate value are top priority

Project Initiation

- Release Plan
  - based on the product backlog and priori7zed items
- Sprint Team
  - Product owner
  - Scrum master
  - The Team

Team Formation

- Introductions and backgrounds
- Team name
- Team room and daily Scrum time/place
- Development process for making product backlog done
- Definition of "Done" for product and Sprint Backlog items
- Rules of development
- Rules of etiquette, and
- Training in conflict resolution

Sprint Planning Meeting

- Part 1, commit to Product Backlog for the next Sprint
  - calculate The Team capacity. Every resource is 100% allocated less 10% for forward looking Product Backlog analysis and 10% for severity 1 issues
  - commit to Product Owner as much backlog as the Team believes it can turn into a "Done" increment in the Sprint

Sprint Planning Meeting

- Part 2, the Team plans out the Sprint
  - self managing teams requiring a tentative plan to start the Sprint
  - tasks that compose this plan are placed in a Sprint Backlog

Sprint / Daily Scrum

- The team gets together for a 15-minute meeting
- Each member answers
  - what have you done on this project since the last Daily Scrum meeting?
  - What do you plan on doing on this project between now and the next Daily Scrum meeting?
  - What impediments stand in the way of you meeting your commitments to this Sprint and this project

Sprint / Daily Scrum

- Do not forget
  - it is the inspect and adapt process control for the Team
  - the 3 questions provide the information the Team needs (inspect) to adjust its work to meet its commitments

Sprint

- What does it mean when a team member says "done"
  - code adheres to the standard
  - is clean
  - has been refactored
  - has been unit tested
  - has been checked in
  - has been built
  - has had a suite of unit tests applied to it

Sprint Review

- Team presents what was developed during the Sprint to the Product Owner and any other stakeholders who want to attend
- Collaborative work session to inspect and adapt:
  - the most current Product Backlog
  - the functionality increment are for inspection,
  - the adaptation is the modified Product Backlog

Scrum Retrospective

- Scrum master encourages the Team to revise, within the Scrum process framework and practices, its development process to make it more effective and enjoyable for the next Sprint

Scrum Advice

- Give it time to get started before expecting big results. It gets better as the team gains experience.
- Tasks for a sprint must be well quantified and achievable within the sprint time period. Determine the sprint time by considering the tasks it contains.
- Tasks for sprints must be assigned to one individual. If the task is shared, give one person the primary responsibility.
- Sprint tasks might include all design-cycle phases. We set goals related to future product releases in addition to current development activity.
- Scrum meetings need not be daily. Two or three times a week works for us.
- The Scrum master must have the skill to run a short, tightly focused meeting.
- Stick to the topics of the sprint. It's very easy to get off topic and extend what was supposed to be a 10 to 15 minute meeting into a half-hour or more.
- Some people are not very good at planning their workload. Sprint goals are an effective tool for keeping people on track and aware of expectations.
- I've noticed an increase in volunteerism within the team. They're taking an interest in each other's tasks and are more ready to help each other out.
- The best part of Scrum meetings has been the problem resolution and clearing of obstacles. The meetings let the team take advantage of the group's experience and ideas.

eXtreme Programming

- XP is what it says, an extreme way of developing software.
  - if a practice is good, then do it all the time.
  - if a practice causes problems with project agility, then don't do it.

eXtreme Programming

- Team, 3-10 programmers + 1 customer
- Iteration, tested and directly useful code
- Requirements, user story, written on index cards
- Estimate development time per story, prio on value
- Dev starts with discussion with expert user

eXtreme Programmign

- Programmers work in pairs
- Unit tests passes at each check-in
- Stand-up meeting daily: Done? Planned? Hinders?
- Iteration review: Well? Improve? => Wall list

XP practices

- Whole Team (Customer Team Member, on-site customer)
- Small releases (Short cycles)
- Continuous Integration
- Test-Driven development (Testing)
- Customer tests (Acceptance Tests, Testing)
- Pair Programming

XP practices

- Collective Code Ownership
- Coding standards
- Sustainable Pace (40-hour week)
- The Planning Game
- Simple Design
- Design Improvement (Refactoring)
- Metaphor

Whole Team

- Everybody involved in the project works together as ONE team.
- Everybody on the team works in the same room. (Open Workspace)
- One member of this team is the customer, or the customer representative.

Small Releases

- The software is frequently released and deployed to the customer.
- The time for each release is planned ahead and are never allowed to slip. The functionality delivered with the release can however be changed right up to the end.
- A typical XP project has a new release every 3 months.
- Each release is then divided into 1-2 week iterations.

Continuous Integration

- Daily build
  - A *working* new version of the complete software is released internally every night
- Continuous build
  - A new version of the complete software is build as soon as some functionality is added, removed or modified

Test-Driven Development

- No single line of code is ever written, without first writing a test that tests it
- All tests are written in a test framework like JUnit so they become fully automated

Customer Tests

- The customer (or the one representing the customer) writes tests that verifies that the program fulfills his/her needs

Pair Programming

- All program code is written by two programmers working together; a programming pair.
- Working in this manner can have a number of positive effects:
  - better code Quality
  - fun way of working
  - skills spreading
  - ...

Collective Code Ownership

- All programmers are responsible for all code
- You can change any code you like, and the minute you check in your code somebody else can change it
- You should not take pride in and responsibility for the quality of the code you written yourself but rather for the complete program

Coding standards

- In order to have a code base that is readable and understandable by everybody the team should use the same coding style

Sustainable Pace

- Work pace should be constant throughout the project and at such a level that people do not drain their energy reserves
- Overtime is not allowed two weeks in a row

Simple design

- Never have a more complex design than is needed for the current state of the implementation
- Make design decisions when you have to, not up front

Design Improvement

- Always try to find ways of improving the design
- Since design is not made up front it needs constant attention in order to not end up with a program looking like a snake pit
- Strive for minimal, simple, comprehensive code

Metaphor

- Try to find one or a few metaphors for your program
- The metaphors should aid in communicating design decisions and intends
- The most well known software metaphor is the desktop metaphor