Software Engineering Project

Morgan Ericsson

morgan@cse.gu.se

@morganericsson

morganericsson

UNIVERSITY OF GOTHENBURG

CHALMERS

Hello

- Morgan Ericsson
  - morgan@cse.gu.se
  - @morganericsson
- Lecturer and examiner

Staff

- Håkan Burden, Lecturer
- Thomas Luvö, Guest lecturer
- Guest lecturer(s)

Textbook

- Online resources and lecture material
- If you want a book, Sommerville's "Software Engineering" (9ed) is a good choice

Course details

- https://github.com/morganericsson/DAT255
  - course material (vc'd)
  - wiki
  - issue tracking
  - previous iterations available as branches ht2014, vt2014, and so on…
- @morganericsson (with #DAT255)
- Further resources will be added during the course...

Practical Details (cont'd)

- Weekly Schedule
  - 1-2 lectures
  - 1 meeting with supervisors
- Presentations w22 (regular lectures)

Examination

- Project (teams)
  - final product
    - artefacts
    - post-mortem experience report
- Brief reflection on group (individual)

Project

- Develop an Android app
  - that does something
    - in teams of approx. 6
- You decide what the app should do and whom you want to work with

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 or BitBucket (private repositories)
  - ...

Outcomes

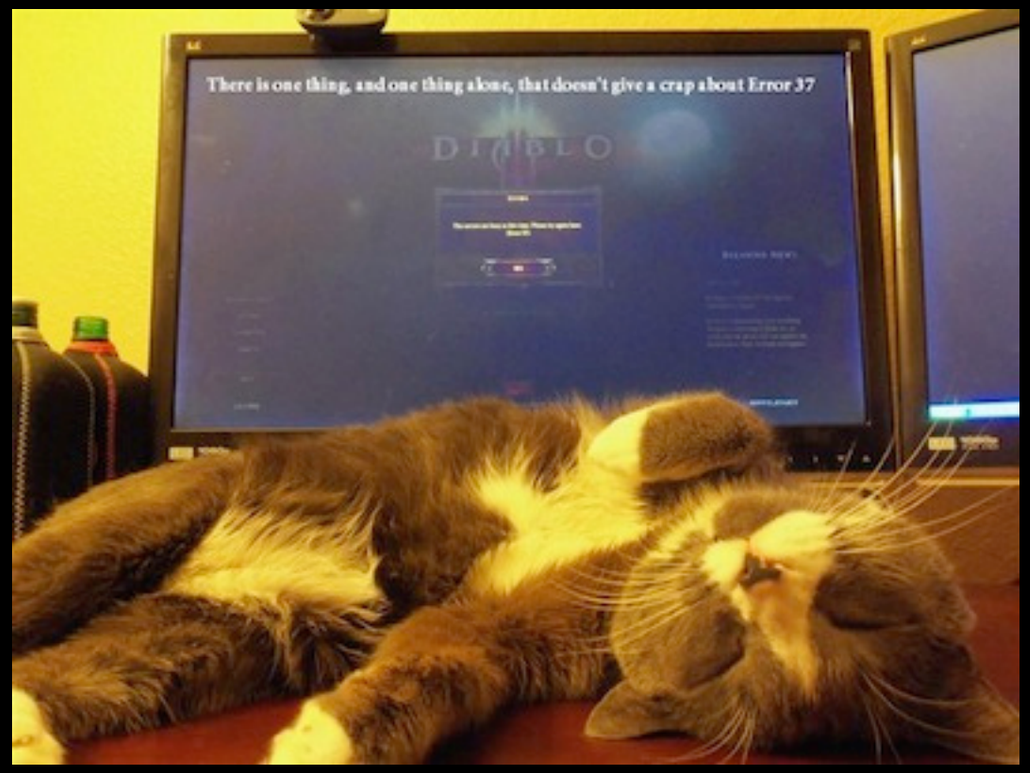
- You will learn a lot, e.g.
  - the software development process
  - useful tools and APIs
- By doing (a lot) and failing (a lot)
- And hopefully have fun while doing it!

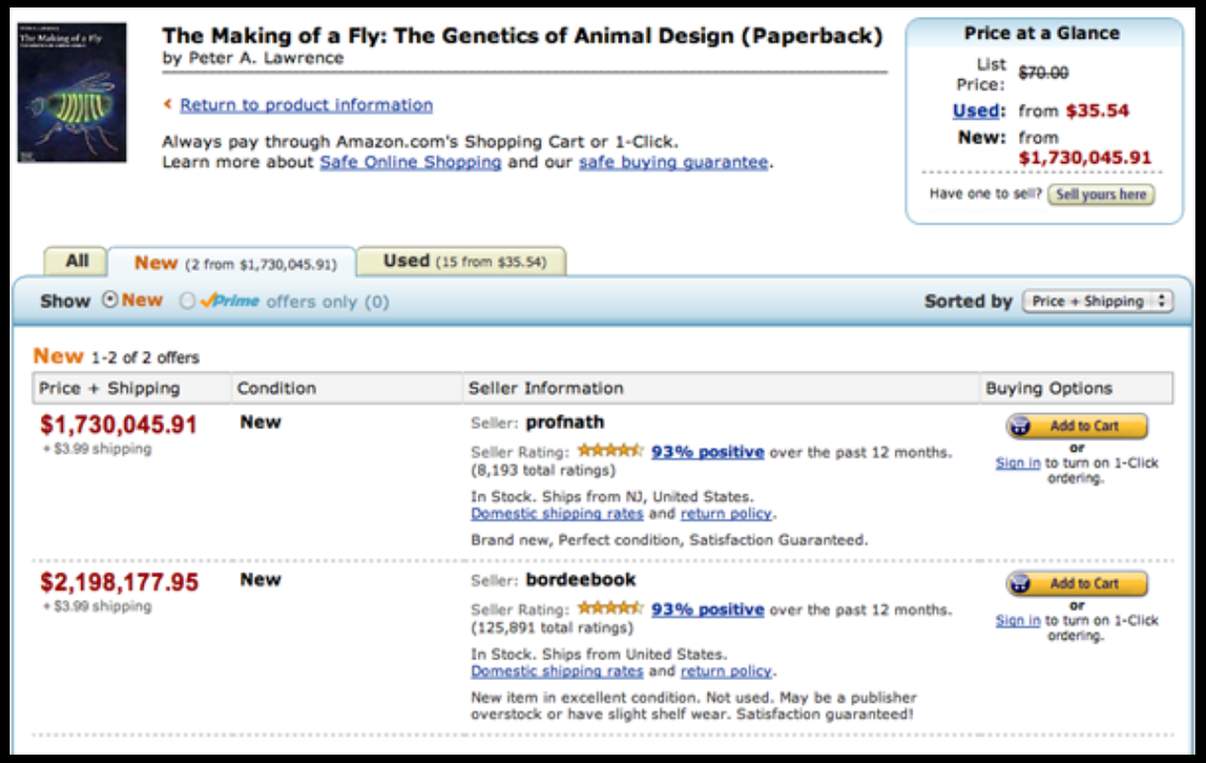
Overall Plan

- Two phases (before and after Easter)
- Phase 1
  - intro to everything
  - you set things up (team, tools, idea, …)
- Phase 2
  - project starts

Week 1

- Intro to course and development process
- Intro to Software Engineering
- You should:
    1. form a team
    2. formulate an idea for an app (vision)
    3. start to write user stories
- If you cannot find a team, matchmaking on Wednesday





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...

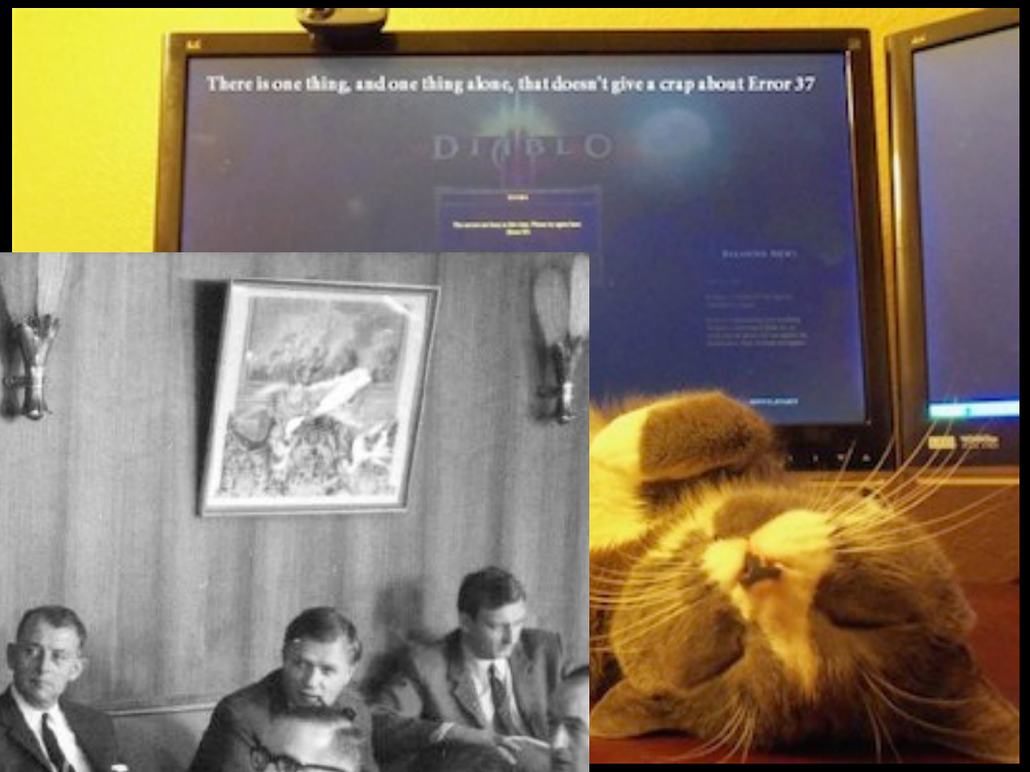
```
Using array [ 0  1  2  3  4 ].
Found 2 at index 2
Found 0 at index 0
Found 3 at index 3
```

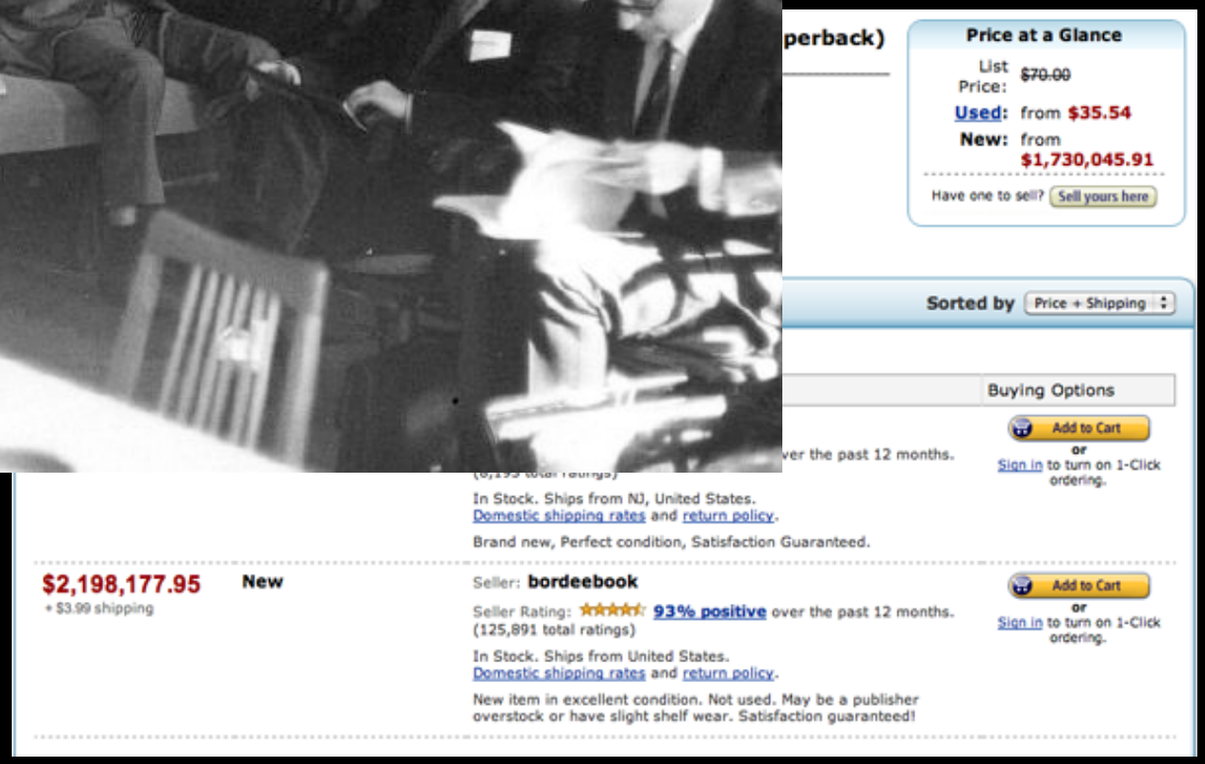
(Seems to work, but...)

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 Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968

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

Requirements

Implementation

Quality Assurance

1. What should I do?

2. Doing it!

3. Did I actually do it?

A sequential model of software development

Five Steps

Requirements

Design

Implementation

Quality Assurance

Evolution

A sequential model of software development

The Five Steps/Phases

- Requirements
  - understanding the problem domain
  - communication between stakeholders
- Design
  - engineer a solution that addresses the requirements
  - technology, algorithms, architecture, interfaces...

The Five Steps/Phases

- Implementation
  - realising the design
    - documentation, configuration, standards, tools, ...
- Quality Assurance
  - ensuring that the implementation meets quality standards
  - testing, analysis, reviews

The Five Steps/Phases

- Evolution
  - fixing problems
  - adding new functionality/address new requirements
  - while retaining existing functionality and code

Software Engineering

Requirements

Architecture

Coding and Detailed Design

Testing

Evolution

Change is Ubiquitous

- Manage
- change
- uncertainty
- quality

Change is Ubiquitous

Design

Requirements

Implementation

Quality Assurance

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

Production vs. Creation

Defined Process control

- Planning and typically pre-study heavy
- Assumes (more) static environment
- Longer iterations
- Change management intensive
- 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-organising teams
- Continuous “customer” interaction
- **NOT UNPLANNED, rather adaptive!!!**

In reality/practice

- Many different
  - processes
  - methodologies
  - practices
  - …

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 do not 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 Programming

- 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 built 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 such as JUnit so they become fully automated

Customer Tests

- The customer (or the one representing the customer) writes tests that verifies that the program fulfils 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 application
- The metaphors should aid in communicating design decisions and intends
- The most well known software metaphor is the desktop metaphor

User Stories

- One or more sentences in the everyday or business language
- Captures what a user does or needs to do as part of his or her job function
- Quick way of handling requirements without formalised requirement documents
- Respond faster to rapidly changing real-world requirements

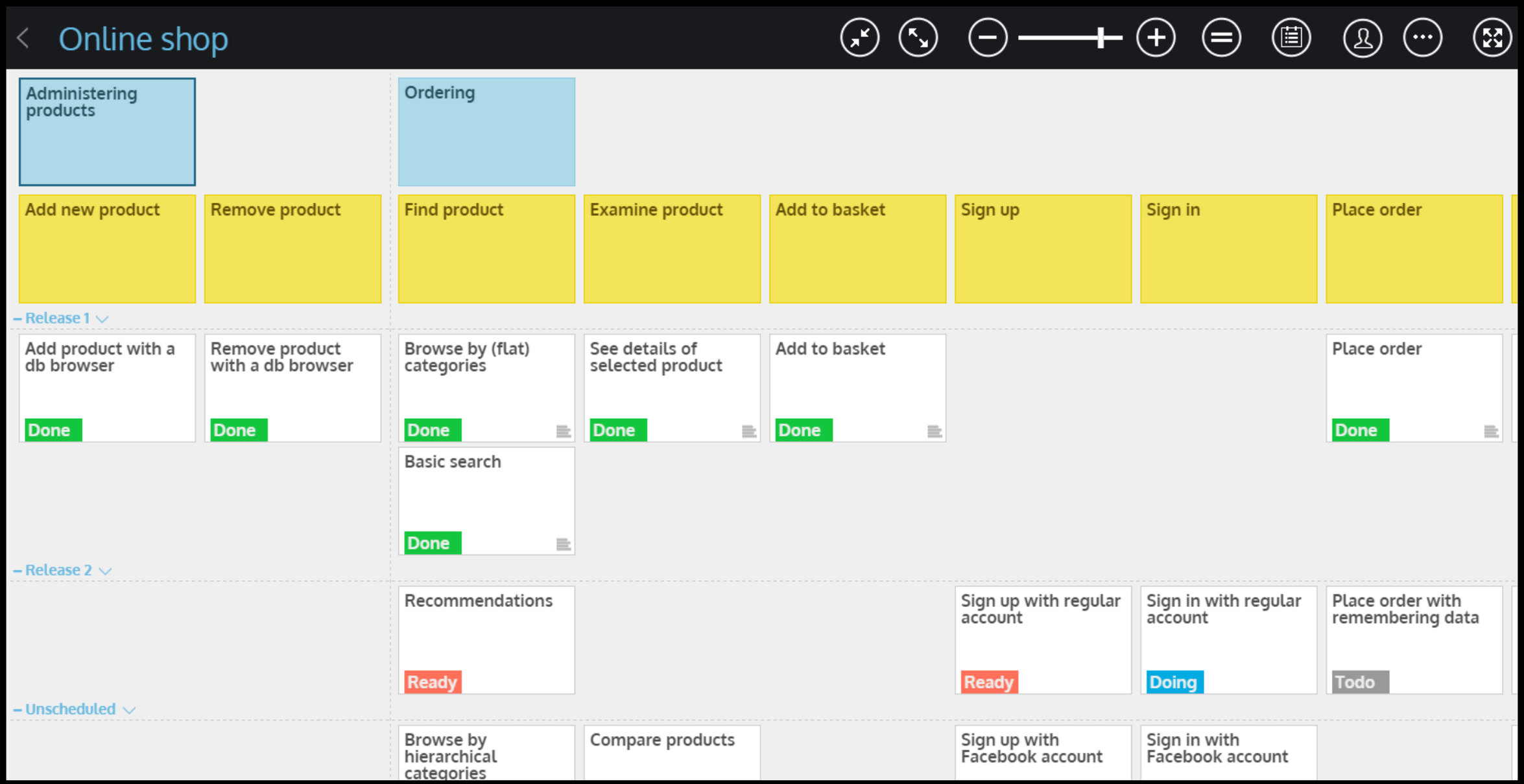
User Stories

- "As a <role>, I want <goal/desire> (so that <benefit>")
- "As <who> <when> <where>, I <what> because <why>."
- *"As a user, I want to search for my customers by their first and last names."*
- *"As a user closing the application, I want to be prompted to save if I have made any change in my data since the last save."*

Benefits

- Represent small chunks of business value that can be implemented in a period of days to weeks
- Needs very little maintenance
- Allows the projects to be broken into small increments
- Are suited to projects where the requirements are volatile or poorly understood. Iterations of discovery drive the refinement process
- Makes it easier to estimate development effort
- Requires close customer contact throughout the project so that the most valued parts of the software get implemented

Story Maps


Online shop
Administering products
Ordering
Add new product
Remove product
Find product
Examine product
Add to basket
Sign up
Sign in
Place order
Release 1
Add product with a db browser
Done
Remove product with a db browser
Done
Browse by (flat) categories
Done
See details of selected product
Done
Add to basket
Done
Place order
Done
Release 2
Basic search
Done
Recommendations
Ready
Sign up with regular account
Ready
Sign in with regular account
Doing
Place order with remembering data
Todo
Unscheduled
Browse by hierarchical categories
Compare products
Sign up with Facebook account
Sign in with Facebook account

Example

Roles

- **Instructor**: Expected to use the website frequently, often once a week. Through the company's telephone sales group, an Instructor frequently places similar orders (for example, 20 copies of the same book). Proficient with the website but usually somewhat nervous around computers. Interested in getting the best prices. Not interested in reviews or other "frills."
- **Experienced Sailor**: Proficient with computers. Expected to order once or twice per quarter, perhaps more often during the summer. Knowledgeable about sailing but usually only for local regions. Very interested in what other sailors say are the best products and the best places to sail.

Stories for Experienced Sailors

- A user can search for books by author, title or ISBN number
- A user can view detailed information on a book. For example, number of pages, publication date and a brief description
- A user can put books into a “shopping cart” and buy them when she is done shopping
- A user can remove books from her cart before completing an order

Stories for Experienced Sailors

- To buy a book the user enters her billing address, the shipping address and credit card information
- A user can rate and review books
- A user can establish an account that remembers shipping and billing information
- A user can edit her account information (credit card, shipping address, billing address and so on)
- A user can put books into a "wish list" that is visible to other site visitors

Stories for Instructors

- A user can view a history of all of his past orders.
- A user can easily re-purchase items when viewing past orders.
- The site always tells a shopper what the last 3 (?) items she viewed are and provides links back to them. (This works even between sessions.)

Assessment

- “A user can search for books by author, title or ISBN number”
- What does that mean? Either or, or any possible combination?
- Fix!
- “A user can do a basic simple search that searches for a word or phrase in both the author and title fields”
- “A user can search for books by entering values in any combination of author, title and ISBN”