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 / Admin
- Thomas Luvö, Guest lecturer
- SICS Guest lecturer(s)
- TA(s)

Student representatives

- ?

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, vt2013, and ht2013
- @morganericsson (with #DAT255)
- Further resources will be added during the course...

Practical Details (cont'd)

- Weekly Schedule
  - 1-2 lectures
  - 1 meeting with supervisors
- Presentation Oct 24 at Lindholmen

Examination

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

Project

- Develop an AGA app (more info Sept 3)
  - that does something
    - in teams of approx. 6
- You decide what the app should do and whom you want to work with (together with TAs)

Environment

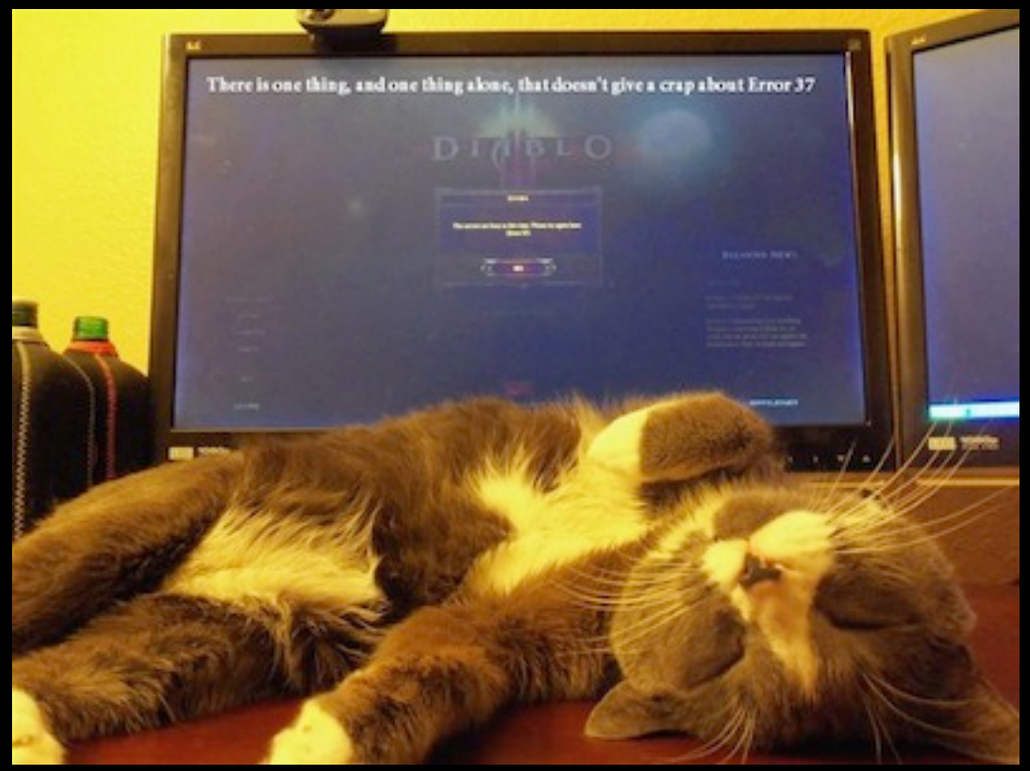
- We strive to create a realistic scenario/ environment
- We rely on a number of real-world services and tools, e.g.
  - AGA SDK
  - GitHub or BitBucket (private repositories)
  - ...

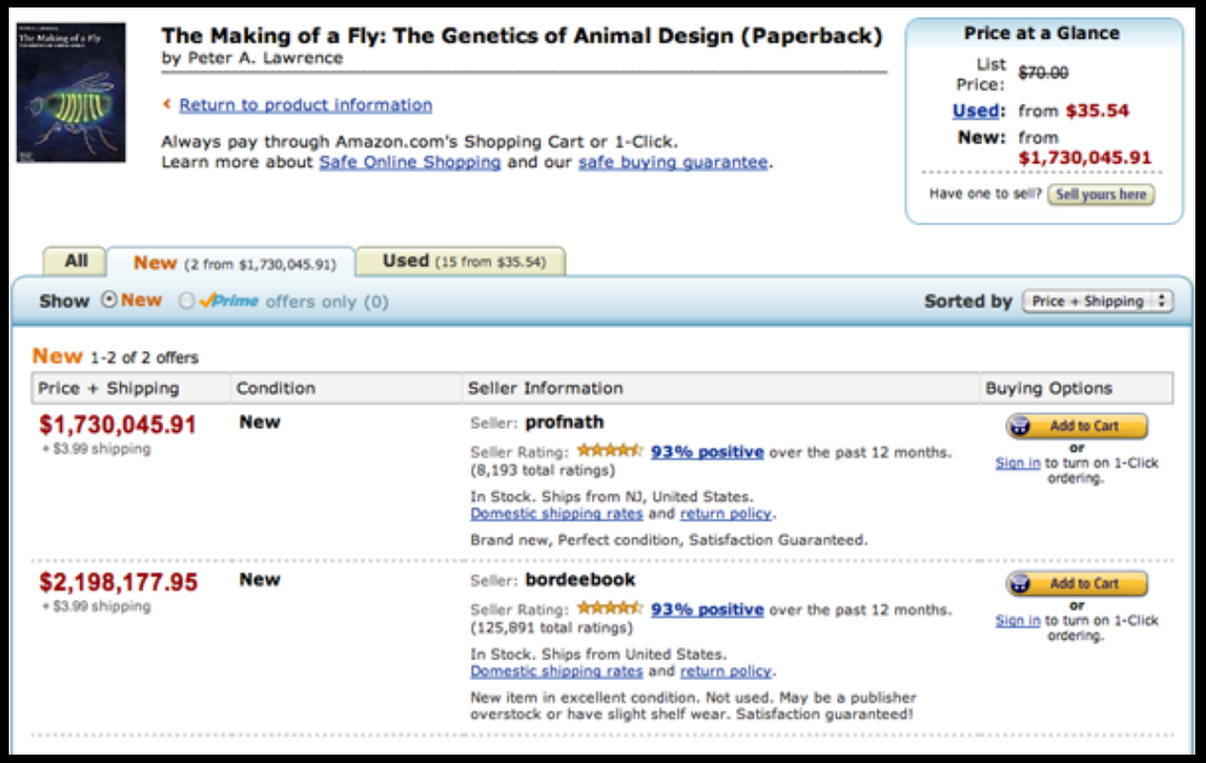
Outcomes

- 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!

Week 1

- Intro to course and development process
- Intro to Software Engineering
- You should:
    1. form a team
    2. formulate three suggestions for an app
    3. submit app ideas to Morgan and Håkan by Sunday (7/9)
- 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: ★★★★½ **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: ★★★★½ **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?

Practical Details

- **Monday**: Course intro and development
- **Wednesday**: Intro SICS/AGA + matchmaking
- **Sunday**: Submit app suggestions