# 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
  - Erik Axelsson
  - Max Witt
  - N.N.

# Textbook

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

# Practical Details

- https://github.com/morganericsson/DAT255
  - course material (vc'd)
  - wiki
    - issue tracking
- @moertech (with #DAT255)
  - updates
- Further resources may be added during the course...

# Practical Details (cont'd)

- Schedule
  - 0-1 lectures per week
  - 0-2 tutorials per week
  - so, 0-3 scheduled activities per week
- Q/A sessions
  - discuss/schedule time with TA

# Examination

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

# Project

- Develop an Android app
  - that does something
    - in teams of approx. 4
- You decide what the app should do and whom you want to work with (together with TA:s)

# 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
  - ..

# Outcomes

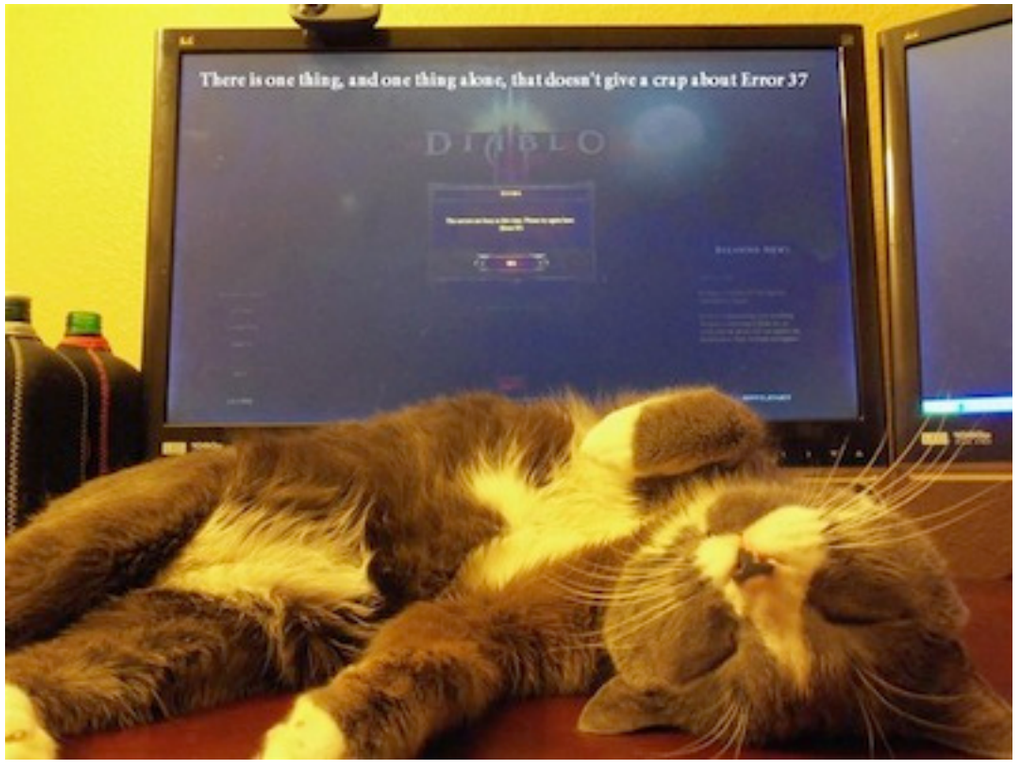
- You will learn a lot, e.g.
  - the software development process
  - useful tools and APIs
- By doing (a lot) and failing (a lot)
- And hopefully have fun while doing it!

# Week 1

- Intro to course and development process
- Intro to Android and tools/services
- You should:
    1. form a team
    2. formulate three suggestions for an app
    3. submit team and app ideas to Morgan no later than April 7
- If you cannot find a team, contact Morgan ASAP (but no sooner!)

# Week 1

- Monday: Course intro
- Wednesday: Intro to Android (virtual lecture)
- 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**<br>Seller Rating: ★★★★★ **93% positive** over the past 12 months. (8,193 total ratings)<br>In Stock. Ships from NJ, United States.<br>Domestic shipping rates and return policy.<br>Brand new, Perfect condition, Satisfaction Guaranteed. | Add to Cart<br>or<br>Sign in to turn on 1-Click ordering. |
| **$2,198,177.95** + $3.99 shipping | **New** | Seller: **bordeebook**<br>Seller Rating: ★★★★★ **93% positive** over the past 12 months. (125,891 total ratings)<br>In Stock. Ships from United States.<br>Domestic shipping rates and return policy.<br>New item in excellent condition. Not used. May be a publisher overstock or have slight shelf wear. Satisfaction guaranteed! | Add to Cart<br>or<br>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."

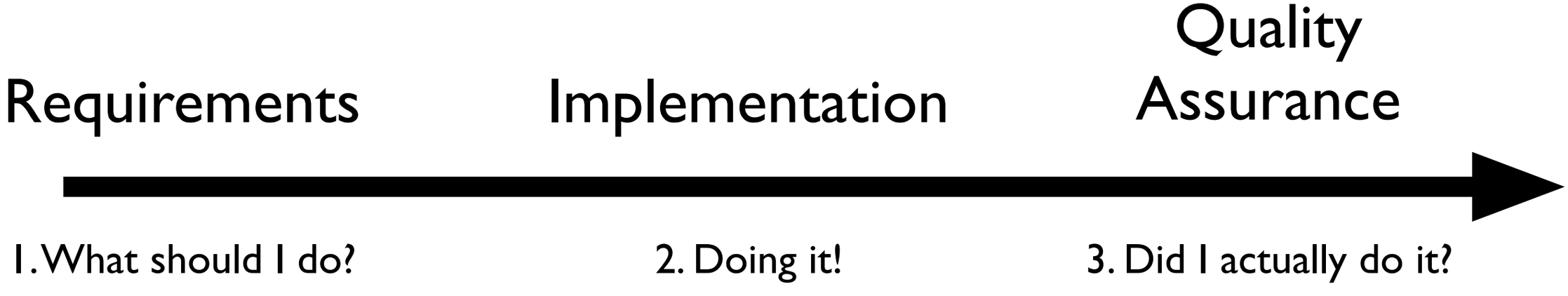
# Software Engineering

- 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”

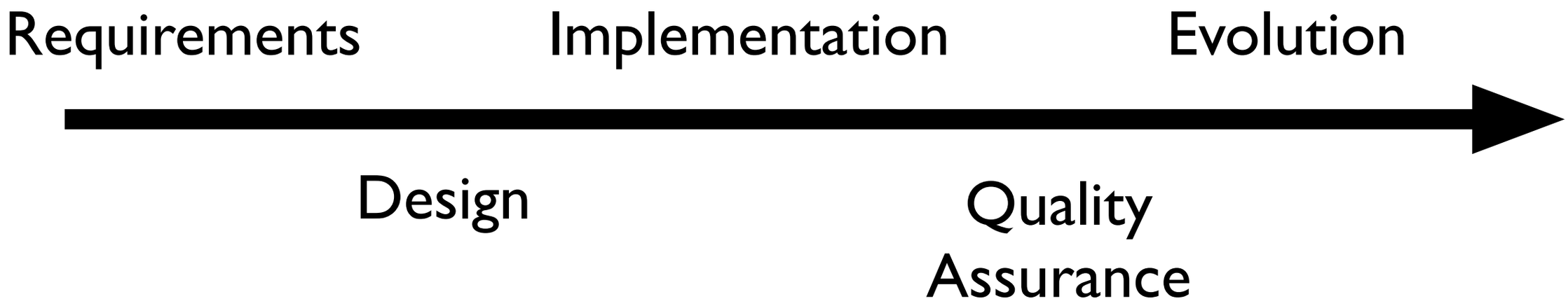
- So, the goal of software engineering is the production of quality software
- However, software is inherently complex
  - the complexity of software systems often exceeds the human intellectual capacity.
- The task of the software development team is to engineer the illusion of simplicity.

# Three Steps Revisited




A sequential model of software development

# Five Steps




A sequential model of software development

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

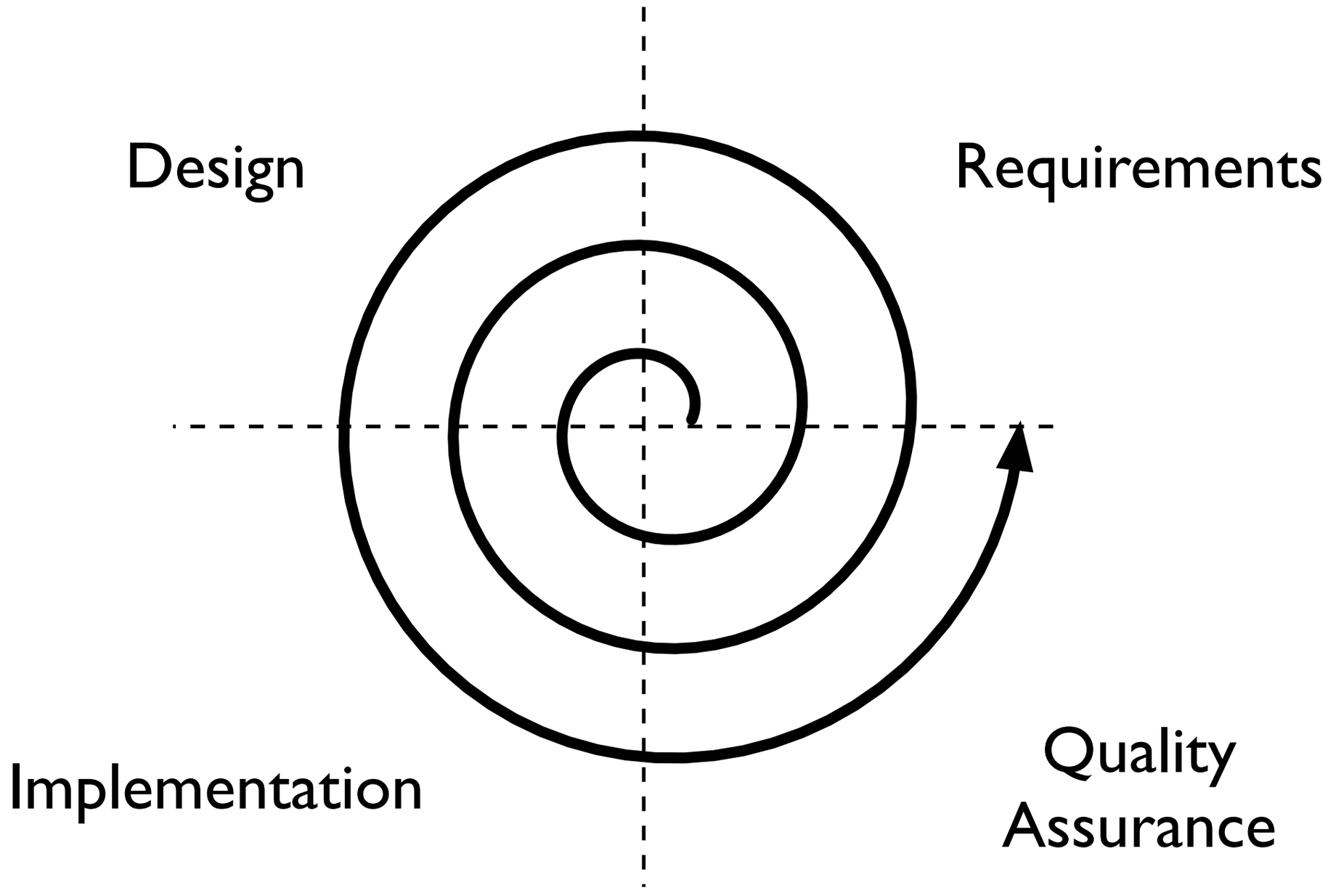
# The Five Steps/Phases

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

# Change is Ubiquitous

- Manage
  - change
  - uncertainty
  - quality

# 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

# Production vs. Creation

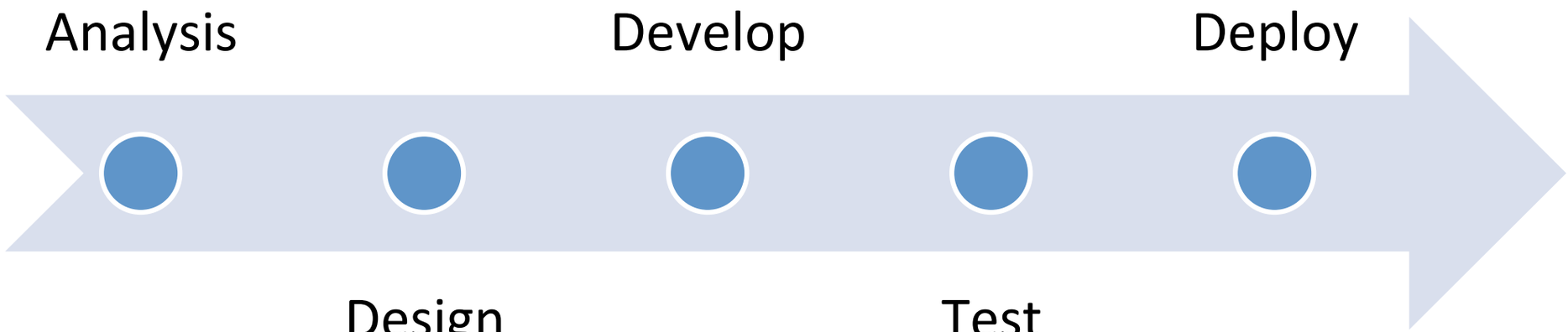
# Defined Process control

- 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)

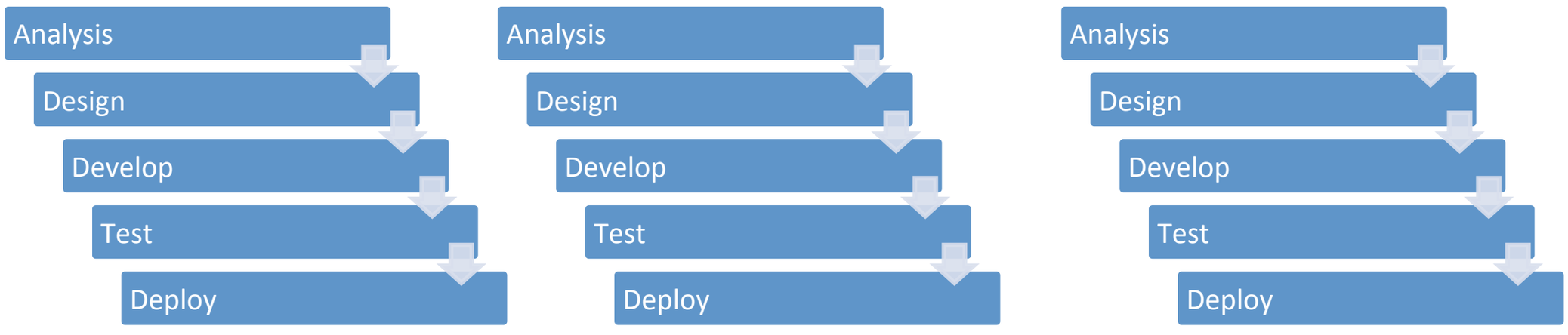
# Empirical Process control

- 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!!!**

# Sequential / Waterfall / Non-agile




# Iterative / Incremental / Agile




Several variants/names, e.g. Scrum, XP, FDD, etc.

# To Be Continued...

- So far, discussion of need for process and what it can look like
- How can this be implemented?
- Week 2: Introduction to Scrum and XP
    - Agile approaches to project management and software development