Perspective
Afterthought is an
Exact Science

Large Scale Programming, 1DL410, autumn 2009
Cons T Åhs
What was this course about?

The official course description:

To write a small program for yourself or as an assignment is one thing.

It is another thing to develop a larger program of high quality with a team.

In this course you won't learn how to program - you should know that - but how to program well.

You learn how to write programs with few errors that can be maintained.

Most of the course is a project, constructing a larger program in a team.

During this process, your program will be inspected and discussed several times.

You will receive constructive criticism so that you can improve the program.

Programming is more than coding: testing, version management, profiling and optimizing, and documentation are part of the course.
Afterthought

‣ By now, you should know a lot more about what is involved in a larger software project.
‣ Constructing program systems is much more than writing code.
‣ Afterthought is an exact science.
  ‣ Good judgement follows from experience.
  ‣ Experience follows from bad judgement.
  ‣ With more experience you will gain insights and do more right from the start, without actually thinking about it.
  ‣ This is what differentiates a seasoned programmer from a novice.
‣ You will hopefully do much better with your next project.
  ‣ You have gained more experience points.
  ‣ The course has hopefully helped you along the way.
‣ What could you have done better?
Important Concepts

‣ Abstraction
  ‣ Separation of concerns; external and internal.
  ‣ The most important (conceptual) tool for writing great programs, both in the short and long run.
  ‣ Correct abstractions are more important than actual tools.
  ‣ Be careful - abstraction is not simple or trivial.
    ‣ Correctly done it is extremely powerful.
    ‣ Incorrectly done it leads to all sorts of nightmares and smells.
  ‣ Abstraction is not about a 1-1 mapping between internal representation and external properties!
  ‣ Allen, Plaugher and Gabriel talks about abstraction - read and read again to take part in their insights. Their perspectives are different, so read all!
  ‣ There is nothing new about this.

‣ Code Standards
  ‣ Seemingly trivial, but leads to easier and more efficient communication within teams. Large companies often enforce one code standard for each language used within the company.
Important Concepts

› Testing and Testability
   ‣ Automated quality assurance.
   ‣ Safety net when refactoring
   ‣ Tested code is reliable
   ‣ Testable code is easier to reuse

› Automation
   ‣ Do not repeatedly do manually what you can automate
   ‣ Computers are very good at repetition
   ‣ Build, test, profile, ..

› Tools
   ‣ all craftsmen use power tools for increased productivity
   ‣ spend time to learn your tools well

› Reviews of design and code
   ‣ get the opinion of your peers
   ‣ embrace critique
   ‣ spread knowledge and learn from each other
Important Concepts

‣ Design smells, code smells
  ‣ code and design changes slowly over time and will start to smell bad if you don’t look after it.
  ‣ you need to be on your watch constantly and keep it check before the smells turns into stenches

‣ Refactoring
  ‣ rewrite code while preserving existing functionality
  ‣ method(s) for combating smells

‣ Profiling and optimisation
  ‣ If possible, change hardware to achieve speed up.
  ‣ If not, measure - don’t guess, before attempting any optimisation.

‣ Scripting
  ‣ tools for small jobs and automation

‣ Formal methods
  ‣ think even more and do it right from the start
  ‣ no testing needed!

tisdag 24 november 2009
Think!

- Think before coding..
  - Design the system, feature or change using high level tools such as UML.
  - Make sure the design is testable.
  - Write the tests and “discover” the interface(s) you need.
  - Write code that fulfils the tests.
  - Refactor to keep the smells out.
- When you start to write code, you are doomed..
  - Code is expensive, both to write, understand and maintain
  - Being more productive means writing less code that is better prepared to change, not writing more code faster.
  - Productivity is not about lines of code/day.

tisdag 24 november 2009
Gabriel talks about ‘Habitability and piecemeal growth’.

- Habitability is about code that is easy to understand and maintain, i.e., great and beautiful code. Just by looking at it you get a good feeling; you feel at home.
- The code is well designed and well written, free from smells and does not resist change.
- Piecemeal growth reflects a “natural” way of developing programs, since change will always happen. Change comes about for different reasons, so we must be prepared for it.
- It is difficult to test for the greatness and beauty of programs
  - it will be apparent when you try to change a program
  - reviews help you to detect this early on
The Future

- Theory can only take you so far; the only way of getting really good at programming in the large is to do it!
  - You can’t learn to drive a car only by reading a book..
- On the other hand, there is a lot of wisdom to be found in books. Some suggestions for future reading
  - The Pragmatic Programmer is an excellent companion.
  - Agile Software Development, Robert C. Martin
  - Clean Code, Robert C. Martin
  - Refactoring, Martin Fowler
  - The Mythical Man Month, Fred Brooks
- Concepts for further study
  - Learn about the craft of writing Compilers
  - Designs Patterns/Anti Patterns
  - UML/design
  - Test Driven Development
Questions?