Course site:

http://www.dcs.bbk.ac.uk/~oded/Tools2011/Tools.html

Assessment

• 50% exam (2 hours)

• 50% Coursework
  • 25% - 4 homework assignments
  • 25% - final project
Future Sessions

Usually be divided into three parts

1. Background and introduction
2. Practice
3. Conclusions and Remarks
Lecturer:

Oded Lachish:

Relevant experience:

- 3 Years at IBM Research Labs
- 3 Year at Motorola (now FreeScale) Semiconductors

You?
This Session

1. Why do you need this module?
2. Software life cycle from a birds eye
3. Environments
4. Gluing the pieces together

... 

Concluding Remarks
Why Do You Need This Module?

To Make you Better!

To Save Them Money!

Them?

Your future employer

Save money?

Training you costs them money
Training *you* costs them *money*

“Really?”

“But I already know how to program in

C, C++, Java, Ruby, Z, Scala, Groovy,…”

Of course you do!

Regretfully in real life

the software development life cycle requires

much more than just programming!
“Why is the software development life cycle (SDLP) so complex?”

Actually the complexity of the manufacturing process depends on the requirements.

The **Financial Services** industry in general has

- Extremely large projects
- Extreme demands on software reliability
- Extreme deadlines
The SDLC Galaxy
Environments (how is software born?)

- **Local**
  Individual Developer

- **Testing**
  QA team Environment

- **Integration**
  Collected work of Individual Developers

- **Stage**
  Load Testing

- **Live**
  Live

- **Live**
“Why are these environments needed?”

Actually they might not be needed.

• The software manufacturing process depends on the goals and for different goals may have a different set of environments

This specific manufacturing process was selected because it is close to that used in the financial sector

“Why is this good for their goals?”

• Hopefully by the end of the term you will know
How do the environments differ?

- Goal
- Teams
- Tools
- Location
- Generality
- ...

Local Environment

- **Goal:** Software Development
- **Teams:** You
- **Tools:** IDE, Version Control, Unit Testing
- **Location:** your work station or a local server
- **Generality:** Mock world
- ...
Integration Environment

- **Goal:** Software Integration, General Testing
- **Teams:** Programmers
- **Tools:** Automated Build tools, Integrations
- **Location:** Dedicated servers
- **Generality:** from pseudo real world to real world
- ...
Testing Environment

- **Goal:** Functional Testing
- **Teams:** Test Team
- **Tools:** Functional testing tools
- **Location:** Dedicated servers
- **Generality:** real world without the “stress”
- ...
Stage Environment

- **Goal**: Load Testing
- **Teams**: Load Team
- **Tools**: Heavy wait simulator
- **Location**: Dedicated servers
- **Generality**: As close to the real world as possible
- ...
Live Environment and Live Environment

• Goal: **Money**

• Teams: Operators

• Tools:

• Location: live servers

• Generality: REAL WORLD!

• ...

17
“So many details already, do you expect us to remember everything”?

Remember: this course is here because it saves them money.

1. The manufacturing process is usually optimized, everything is done for a reason. So it is easier to remember.

2. This knowledge may come handy in job interviews, (motivation helps memory).

Problem: A good interviewer sees beyond the buzz words.
Problem: “A good interviewer sees beyond the buzz words”

Solutions:

1. Try to use as much of this process as possible in your masters project – this may seem an overkill, but the goal is to put content behind your buzzwords (don’t overdo)

2. Same as (1) but with a project of your own. Such projects show that you are able to work autonomously, some employers find this to be a very important quality (note that this is not the opposite of team player)

3. Contribute to an open source project
One critical detail to remember before we go on!

Automation

As much as possible.

Why?

• Saves money

• Lower probability for human error
SDLC

The Code Entry Point
The Code Entry Point

The development cycle does not start with code, except maybe in *extreme* cases.

We shall discuss this a bit later in the module and a lot more in the OODP module.
How do you develop software?

Me?

I write the code, all of it compile

AND IT WORKS

EVERY SINGLE TIME

(that is why I teach)

You?
Test Driven Development (TDD)

Philosophy

• Untested code is “not code”

• Short development cycle

• Testing is understanding
New feature, write a test to make sure you understand

Test should fail, since there is no implementation
TDD benefits

• Less Debugging (debugging is tedious and expensive)

• Small steps – easier to backtrack (less debugging)

• Small pieces drive developers towards modularity

• Bugs caught earlier

• Testing is understanding
TDD vulnerabilities

• Developer writing tests for his own code

• Checking private variables may require dedicated “hacks” that may remain in the code.

• More code

• Longer time to write

• Over reliance on unit testing
TDD implementation

Wish List

• Easy to upgrade code (refactor)
• Dedicated testing tools
• Friendly version control

Your bosses wish list

• Automated code style enforcement
Tool Types

- IDE – Integrated Development Environment
- Version Control
- Tools for testing by simulation
- Code coverage
- Mock Libraries
- Static Analysis
- Debugging
- Profiling
- Compilers
- Interpreters
- Build Tools
IDE – Integrated Development Environment

• A long long time ago all the tools were separate.
• Since then things have changed.
• The IDE integrates many (but not all) of them into one big GUI.
• The goal is to save the time wasted on switching between the tools.
• Note that achieving such a goal is highly non-trivial
Version Control

• A big project can easily have thousands of different files.
• Many of these files may be the responsibility of more than one person
• There might even be different versions of the same project

This may cause a variety of issues:
• Contradicting changes
• Wrong selection of files

Version control is there to resolve such issues.
Unit Testing

• When describing TDD we mentioned the emphasis on Testing every little piece of code.

• One way to do this is to simply write a program that uses the code you have written.

Issue: this program is not an active part of the final product

Solution: Automate the process of writing such programs – unit test are created in a semi automatic way
Mock Libraries

- Semi automatically generating unit test is great.
- But, what if testing requires access to things that have not been written yet, for example: Databases, Proxy Servers...

Mock libraries contain pre-prepared objects that mock the behaviour of such objects.
Code Coverage

How do you know you have tested enough.
(At least in theory it seems you never know)
However one popular metric is code coverage.

The idea is to check for example

• All lines of code were executed
• All conditions in the code evaluated to every possible value

Naturally this process is automated.
Static Program Analysis

Analyzing the program without executing it

Tool types:

• **Check coding style**
• Formal verification – prove mathematically that your code has no bugs!
• Reverse engineer
Debugging

• A test has failed now it is time to find out why.
  (ideally just go back to the latest version that worked and rewrite)

• During the debugging process one wants to stop the program at certain lines, query values of variables etc.

  Naturally this process is automated.
Profiling

• All the tests passed Great?

• Program execution takes forever %^£&”!

• Where is all the time wasted?

Naturally this process is automated.
Build

• We have been talking about testing
• However it is not necessarily trivial to get there

• Building a running program may require more than just pressing a button.

Build tools are the in order to simplify and automate
Integration

Where individuals become a team
Integration

Where all the code goes to (and also the unit tests)

Here things can really go wrong

When things do go wrong we want to minimize the damage
Continuous Integration

Integrate code as soon as possible. Why?

Each time only a small portion of code is added or changed. As a result

- Problems are detected earlier
- Easier to find the problem
Continuous Integration

How?
Automatically of course.

• Once a programmer finishes working on a piece of code, he uses the version control to incorporate it into a repository.
• The repository is automatically sampled by a continuous integration server. Once the server detects change, it goes into action.
Continuous Integration Server

Change detected.

• Build project (usually on dedicated servers)
• Run unit tests
• Run integration tests
• Run ...
• Send reports to relevant team members by e-mail

All automatically
(sometimes doing a job that required to people)
Choosing the Tools
Open Source Vs Proprietary

- “Cheap”
- “Un – Reliable”
- “Support”
- Open Source
- Flexible
- *It’s your fault*

- “Expensive”
- “Reliable”
- Dedicated Support
- Source Hidden
- Tailored
- *Some one to blame*
Open Source Vs Proprietary

- GUI might not be so nice
- **Open Source** enables one GIANT integrated design environment

- Nice GUI
- **Starting to integrate with open source IDEs**

“Support”
List of tools in module (not final)

- Eclipse Indigo - IDE
- Netbeans - IDE
- GIT - Version Control
- JUnit 4 - Unit Testing
- Corbetura - Code Coverage
- PMD - Code Style
- CheckStyle - Code Style
- FindBugs - Code Style
- Jupiter - Code Style
- Bugzilla - Bug-Tracking System
- Trac - Issue tracking system
- Maven - software project management
- Ant - build of Java applications
- Jenkins - Continuous integration server
“Epilogue”
Module Goals

• Get the students to become acquainted with the tools used in the SDLC

Emphasis

• Develop students confidence to tour the SDLC galaxy

Remark: The quantity of material covered is secondary

(the SDLC galaxy is GIANT and constantly changing, hence the ability to acquire new knowledge and adapt is more important the actual knowledge acquired)
Concluding Remarks

It is important to remember that what is taught in this course is only a small portion of a huge evolving galaxy.

Different companies may have extremely different software life cycles (even the same company)

Another goal of this module is to make you aware of this.