Software Development Tools

IDEs
Revision Control
Using GitLab and Git
Integrated Development Environments

- A rich context for software development
  - Examples: Eclipse, IntelliJ, VisualStudio, XCode
- Syntax highlighting, continuous compilation, testing, debugging, packaging
- Powerful refactoring capabilities
Revision Control

- Indispensable software engineering tool
- Solitary work
  - Personal audit trail and time machine
  - Establish when bug was introduced
  - Fearlessly explore new ideas (roll back if no good)
- Teamwork
  - Concurrently develop
  - Share work coherently
Git

• Distributed version control system
  – hg, git, others (conceptually very similar)

• Contrast with centralized version control
  – cvs, svn, others

We will focus on distributed version control systems and not discuss centralized version control any further.
**Git & GitLab**

**Introduction to Software Systems 1110/1140/6710**

**Git & GitLab Lab**

**Purpose**
Task 1: In this lab, you should fork the lab 1 repository to your own account. This is an essential step for the tasks in this lab. Forking allows you to make your own changes to the code without affecting the original repository. Please follow the instructions below:

1. **Fork the GitLab repository.**
   - Click on the fork button on the top right of the repository.
   - Fork your own account.

2. **Clone your forked repository.**
   - Run the command `git clone https://<your_fork>.git` in your terminal.
   - Replace `<your_fork>` with your actual forked repository URL.

3. **Push your changes.**
   - Open the repository in your GitLab account.
   - Make changes to the code as per the instructions.
   - Click on the push button to push your changes to your forked repository.

**Tasks**

1. **Fork the GitLab repository.**
   - Follow the instructions above to fork the repository.
2. **Clone your forked repository.**
   - Follow the instructions above to clone your forked repository.
3. **Push your changes.**
   - Follow the instructions above to push your changes.

**Purpose**
Task 2: In this lab, you should clone the forked repository to your local machine. This is an essential step for the tasks in this lab. Cloning allows you to work on the code locally without affecting the original repository. Please follow the instructions below:

1. **Clone your forked repository locally.**
   - Run the command `git clone https://<your_fork>.git` in your terminal.
   - Replace `<your_fork>` with your actual forked repository URL.

2. **Checkout the branch.**
   - Run the command `git checkout <branch_name>` in your terminal.
   - Replace `<branch_name>` with the name of the branch you want to work on.

3. **Update your code.**
   - Pull the latest changes from the remote repository.
   - Run the command `git pull origin <branch_name>` in your terminal.
   - Replace `origin` with your GitLab repository.

4. **Push your changes.**
   - Commit your changes locally.
   - Run the command `git add .`
   - Run the command `git commit -m "Your commit message"`
   - Replace `Your commit message` with your actual commit message.
   - Push your changes to the remote repository.
   - Run the command `git push origin <branch_name>` in your terminal.
   - Replace `origin` with your GitLab repository.

**Tasks**

1. **Clone your forked repository locally.**
   - Follow the instructions above to clone your forked repository locally.
2. **Checkout the branch.**
   - Follow the instructions above to checkout the branch.
3. **Update your code.**
   - Follow the instructions above to update your code.
4. **Push your changes.**
   - Follow the instructions above to push your changes.
IntelliJ Git Integration

- Create a new repository:
  - VCS->Import into Version Control->Create Git Repository…
- Clone an existing repository:
  - VCS->Checkout from Version Control->Git…
- Other operations:
  - VCS
  - VCS->Git
  - right mouse click -> Git
Revision Control

Git
Git Concepts

- Commit (noun)
- Staging (IntelliJ allows you to more or less ignore this, so we will)
  ✔ Commit (atomically commit changes to your local repo)
  ✔ Push (push outstanding local changes to a remote repo)
  ✔ Pull (pull new changes from a remote repo)
  ✔ Update (update your working version)
- Merge
- Reset and Revert
Git Commits

Captures a set of changes (including modifications, additions and deletions) that may span multiple files.

- Globally unique commit ID (large hexadecimal number)
- Parent – child relationship (based on changeset ID)
  - Single parent, single child is simple case
  - Multiple children indicates a *branch*
  - Multiple parents indicates a *merge*
- A push sends commits, a pull gets commits
- Commits are usually never deleted
A Little More on Update

Update will by default take you to the “HEAD” (the most recent known commit).

You can, however, “update” to any particular revision, moving yourself back and forward in time. To do this, you need to specify the revision.

In IntelliJ you can do this by double-clicking on the revision (VCS -> Git -> Show History, then select the revision)
Branches and Merging

A **branch** occurs when a commit has more than one *child*. A **merge** is special commit with two *parents* (thus uniting branches). If branches are *conflicting* (changes to the same file), those conflicts must be **resolved** before merging.
Amend Reset and Revert

You can amend a commit message with `amend`
You can reset your local state to a particular commit (throwing away un-pushed changes whether committed or) with `reset`.
You can also `revert` any particular commit. This amounts to applying a counteracting commit.
When All Else Fails

This is Git. It tracks collaborative work on projects through a beautiful distributed graph theory tree model.

Cool. How do we use it?

No idea. Just memorize these shell commands and type them to sync up. If you get errors, save your work elsewhere, delete the project, and download a fresh copy.
Software Development Teams

Code of Conduct

You have two primary responsibilities:

- **Promote** an inclusive, collaborative learning environment.
- **Take action** when others do not.

**Professionally, we adhere to ACM’s Code of Ethics.** More broadly, a course like COMP1110 involves reflection, collaboration, and communication. Computer science has a checkered history with respect to inclusion – in corporate environments, in our classrooms, and in the products we create. We strive to promote characteristics of transparency and inclusivity that reflect what we hope our field becomes (and not necessarily what it has been or is now).

Above all, **be kind.**

**We reject behaviour that strays into harassment, no matter how mild.** Harassment refers to offensive verbal or written comments in reference to gender, sexual orientation, disability, physical appearance, race, or religion; sexual images in public spaces; deliberate intimidation, stalking, following, harassing photography or recording, sustained disruption of class meetings, inappropriate physical contact, and unwelcome sexual attention.

If you feel someone is violating these principles (for example, with a joke that could be interpreted as sexist, racist, or exclusionary), **it is your responsibility to speak up!** If the behaviour persists, send a private message to your course convener to explain the situation. We will preserve your anonymity.

(This code of conduct was developed by Evan Peck of Bucknell University. Portions of this code of conduct are adapted from Dr. Lorena A. Barba)
Test Driven Development

Test-Driven Development (TDD)
JUnit
Test Driven Development (TDD)

TDD “red, green, refactor”
1. Create test that defines new requirements
2. Ensure test fails
3. Write code to support new requirement
4. Run tests to ensure code is correct
5. Then refactor and improve
6. Repeat

Key element of agile programming
JUnit

Unit testing for Java

• Developed by Kent Beck
  – Father of extreme programming movement

• Integrated into IntelliJ

• Useful for:
  – TDD (Test driven development)
  – Bug isolation and regression testing
    • Precisely identify the bug with a unit test
    • Use test to ensure that the bug is not reintroduced
JUnit

- Methods marked with `@Test` will be tested
- When JUnit is called on a class, all tests are run and a report is generated *(a failed test does not stop execution of subsequent tests).*
- JUnit has a rich set of annotations that can be used to configure the testing environment, including:
  - `@Test`, `@Ignore`, `@Before`, `@BeforeClass`, `@After`, `@AfterClass`
- JUnit can check that an exception is thrown if that is expected in a certain case
  - `@Test(expected = ArithmeticException)`
What is Software Engineering?
Does Software Engineering Matter?
Software Engineering

Very roughly:

Software engineering is concerned with the efficient and timely delivery of software that meets stated requirements.
Software Engineering

Software project success rates:

Success 10-30%
Challenged 50-75%
Failed 10-30%

The error which ultimately led to the destruction of the Ariane 5 launcher about 40 seconds after lift off on its maiden flight was clearly identified in the report of the investigating committee [1]: a program segment for converting a floating point number to a signed 16 bit integer was executed with an input data value outside the range representable by a signed 16 bit integer. This run time error (out of range, overflow), which arose in both the active and the backup computers at about the same time, was detected and both computers shut themselves down. This resulted in the total loss of attitude control. The Ariane 5 turned uncontrollably and aerodynamic forces broke the vehicle apart. This breakup was detected by an on-board monitor which ignited the explosive charges to destroy the vehicle in the air. Ironically, the result of this format conversion was no longer needed after lift off.

Robert L. Baber 2002
[...] an audit report by the consulting company KPMG into the status of the payroll system indicates that it will cost another A$220.5 million—on top of the A$311 million already spent—to fix nine priority items that prevent the payroll of the 85,000 or so Queensland Health employees from being calculated without massive manual intervention.

[...] back in 2008, the original cost of the payroll system development was pegged at A$6.19 million (fixed price), which has steadily grown as problems such as the massive overpayment or underpayment of employee salaries ran rampant.

Robert Charette, IEEE Spectrum 2012
Nasdaq OMX Group Inc [...] blamed “poor design” in the software it uses for driving auctions in initial public offerings. Computer systems used to establish the opening price were overwhelmed by order cancellations and updates [...]. Nasdaq’s systems fell into a “loop” that kept the second-largest U.S. stock venue operator from opening the shares on time following the $16 billion deal.

“It’s amazing that both Bats and Nasdaq unfortunately failed in an inglorious way,” William Karsh, the former chief operating officer at Direct Edge Holdings LLC, an exchange operator that competes with Nasdaq, said in a telephone interview yesterday. “It proves that technology isn’t infallible. There are so many moving parts that things can go wrong. That’s the lesson we learn.”

Bloomberg 22/5/2012
[...] the Patriot’s failure was at least in part caused by a software flaw. Hitting the incoming Iraqi Scud missile was within the capability of the Patriot system, yet it missed. [...] Yet to characterize this failure as a bug or programming lapse misses a larger point.

This failure can be seen as boneheaded software management. The case can be made that the problem is better traced to a framework flaw. [...] More to the point, a suitable test framework would have detected the flaw, using the same compilers used in the Patriot and without impugning the skills of the Patriot developer team, who may we have excelled in other aspects of that complex software project.

Mark Underwood, Technorati 1/11/2009
Software Engineering: Human Costs

AF447, 1/6/2009, 228 Dead

<table>
<thead>
<tr>
<th>Time</th>
<th>Call</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>02:12:27</td>
<td>PNF</td>
<td>You are climbing.</td>
</tr>
<tr>
<td></td>
<td>VS</td>
<td>Stai Stai</td>
</tr>
<tr>
<td>02:12:30</td>
<td>PNF</td>
<td>You are descending, descending descending</td>
</tr>
<tr>
<td>02:12:32</td>
<td>PIC</td>
<td>No, you are climbing</td>
</tr>
<tr>
<td>02:12:42</td>
<td>PF</td>
<td>OK, we are in TOGA</td>
</tr>
<tr>
<td>02:12:42</td>
<td>PF</td>
<td>On the altitude where are we?</td>
</tr>
<tr>
<td>02:12:44</td>
<td>PIC</td>
<td>this is not possible</td>
</tr>
<tr>
<td>02:12:45</td>
<td>PF</td>
<td>On attitude we are where?</td>
</tr>
<tr>
<td>02:12:45</td>
<td>PNF</td>
<td>What do you mean on attitude?</td>
</tr>
<tr>
<td>02:12:45</td>
<td>PF</td>
<td>Yes, yes, yes, I am descending there, no?</td>
</tr>
<tr>
<td>02:12:45</td>
<td>PNF</td>
<td>Yes, you are descending.</td>
</tr>
<tr>
<td>02:12:45</td>
<td>PIC</td>
<td>Hey, you are in..... put the wings level,</td>
</tr>
<tr>
<td>02:12:45</td>
<td>PNF</td>
<td>Put the wings level</td>
</tr>
<tr>
<td>02:12:45</td>
<td>PF</td>
<td>That is what I am trying to do</td>
</tr>
<tr>
<td>02:12:46</td>
<td>PIC</td>
<td>Put the wings level</td>
</tr>
<tr>
<td>02:12:59</td>
<td>PF</td>
<td>I am at the limit of, with the warping</td>
</tr>
<tr>
<td>02:13:25</td>
<td>PF</td>
<td>What, how is it that we are continuing to descend at the limit there?</td>
</tr>
<tr>
<td>02:13:28</td>
<td>PNF</td>
<td>Try to find what you can do with the controls up there, The primaries</td>
</tr>
<tr>
<td>02:13:32</td>
<td>PF</td>
<td>At level 100</td>
</tr>
<tr>
<td>02:13:36</td>
<td>PF</td>
<td>9000 ft</td>
</tr>
<tr>
<td>02:13:38</td>
<td>PIC</td>
<td>Carefull with the rudder!</td>
</tr>
</tbody>
</table>

Transcripts from BAE, via avherald
Software Engineering: Human Costs

AF447, 1/6/2009, 228 Dead

Transcripts from BAE, via avherald

02:13:38 PIC  Carefull with the rudder!
02:13:39 PNF  Climb, climb. Climb, climb
02:13:40 PF    But I am at the limit of the nose since a while
02:13:41 PIC  No, no, no, don’t climb
02:13:45 PNF  So descend
02:13:45 PF    So, give me the controls, to me the controls.
02:13:45 PNF  Go ahead, you have the controls, we are still on TOGA
02:14:05 PIC  Careful, you are nose high (cabrea?)
02:14:10 PNF  I am nose high?
02:14:10 PF    Well, we need to, we are at 4000 ft
02:14:18 PIC  Go, Pull
02:14:18 PF    Go, Pull pulpu!
02:14:26 PIC  Ten degrees pitch

- The lack of a clear display in the cockpit of the airspeed inconsistencies identified by the computers;
- The crew not taking into account the stall warning, which could have been due to:
  -- A failure to identify the aural warning, due to low exposure time in training to stall phenomena, stall warnings and buffet,
  -- The appearance at the beginning of the event of transient warnings that could be considered as spurious,
  -- The absence of any visual information to confirm the approach-to-stall after the loss of the limit speeds,
  -- The possible confusion with an overspeed situation in which buffet is also considered as a symptom
  -- Flight Director indications that may led the crew to believe that their actions were appropriate, even though they were not
  -- The difficulty in recognizing and understanding the implications of a reconfiguration in alternate law with no angle of attack protection.
Key Facets of Software Engineering

• Requirements
• Design
• Implementation
• Testing
• Quality
• Maintenance
• Configuration Management
COMP1510
Software Development Models

Big Design Up Front
Waterfall
Spiral
Agile
Formal Methods
The Waterfall Model and “Big Design Up Front”
Benington 1956, Royce 1979, et al

These emphasize getting design absolutely right before progressing the development. Waterfall applies this to all phases: each phase must be finalized before moving to the next.

Waterfall stages:
- Requirements
- Design
- Implementation
- Verification
- Maintenance
The Spiral Model
Barry Boehm, 1986

This model is iterative, unlike the waterfall model.

Each iteration includes steps like those in the waterfall model. The spiral model is based on prototyping and iterative refinement.
Agile Development
Beck et al 2001

value-driven rather than plan-driven
dynamic rather than static

Agile Manifesto
We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Beck et al 2001
Formal Methods

Rigorous mathematical approach to verifying correctness of implementation.

Requires

- Formal specification
- Verification of implementation
- Theorem proving assistance (interactive theorem provers)
Which Approach?

*Often viewed as a religious question…*

Here’s Boehm & Turner’s take:

<table>
<thead>
<tr>
<th>Agile home ground</th>
<th>Plan-driven home ground</th>
<th>Formal methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low criticality</td>
<td>High criticality</td>
<td>Extreme criticality</td>
</tr>
<tr>
<td>Senior developers</td>
<td>Junior developers</td>
<td>Senior developers</td>
</tr>
<tr>
<td>Requirements change often</td>
<td>Requirements do not change often</td>
<td>Limited requirements, limited features</td>
</tr>
<tr>
<td>Small number of developers</td>
<td>Large number of developers</td>
<td>Requirements that can be modeled</td>
</tr>
<tr>
<td>Culture that responds to change</td>
<td>Culture that demands order</td>
<td>Extreme quality</td>
</tr>
</tbody>
</table>
The Mythical Man Month,
Fred Brooks, 1975

*Brooks’ law: “Adding manpower to a late project makes it later.”*

Brooks’ experience leading the development of IBM’s OS/360.

Much of what Brooks describes are what we now call ‘*anti-patterns*’.

- Adding manpower to a late project makes it later
  - Large complex projects are communications-intensive
  - Adding new people is very costly in terms of communications
  - The communications overhead will eventually dominate

- Second system effect
  - Second implementation is dangerous
  - Tend to want to incorporate all the ideas discarded as impractical

- Scheduling
  - “Q: How does a project get one year late? A: One day at a time!”
No Silver Bullet
Fred Brooks, 1986

“building software will always be hard. There is inherently no silver bullet.”

“Software entities are more complex for their size than perhaps any other human construct.”

“Despite progress in restricting and simplifying software structures, they remain inherently unvisualizable, and thus do not permit the mind to use some of its most powerful conceptual tools”
No Silver Bullet,
Fred Brooks, 1986

**Accidental Complexity**: artifacts of production of s/w

**Essential Complexity**: *inherent* in the nature of the s/w

- difficulty in communicating among team: *product flaws etc*
- difficulty in enumerating and understanding all states: *unreliability*
- difficulty of invoking function: *software is hard to use*
- difficulty of extending programs: *unanticipated states, security flaws*
Design Patterns: Elements of Reusable OO Software
Gamma, Helm, Johnson, Vlissides, 1994 (aka “The Gang of Four”)

A long history of using *patterns* in engineering.

The GoF identify 23 *software design patterns*:

- **5 creational** (create objects)
  - *Prototype* creates an object by cloning an existing one.

- **7 Structural** (describe object composition)
  - *Proxy* functions as an interface to something else.

- **11 Behavioral** (describe communication between objects)
  - *Visitor* separates algorithm from structure.
The Five Orders of Ignorance
Phillip Armour, 2000

“the hard part of building systems is not building them, it’s knowing what to build”

“If we view systems development as the acquisition of knowledge, we can also view it as the reduction or elimination of ignorance.”
The Five Orders of Ignorance
Phillip Armour, 2000

0th Order Ignorance: Lack of Ignorance. I have 0OI when I (probably) know something.

1st Order Ignorance: Lack of Knowledge. I have 1OI when I don't know something. With 1OI we have the question in a well-factored form.

2nd Order Ignorance: Lack of Awareness. I have 2OI when I don't know that I don't know something.

3rd Order Ignorance: Lack of Process. I have 3OI when I don't know a suitably efficient way to find out I don't know that I don't know something.

4th Order Ignorance: Meta-ignorance. I have 4OI when I don't know about the Five Orders of Ignorance.