IDEs
Revision Control
Using Gitlab and Git

ANU – School of Computing – Structured Programming 1110 / 1140 / 6710

Integrated Development Environments

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

Revision Control

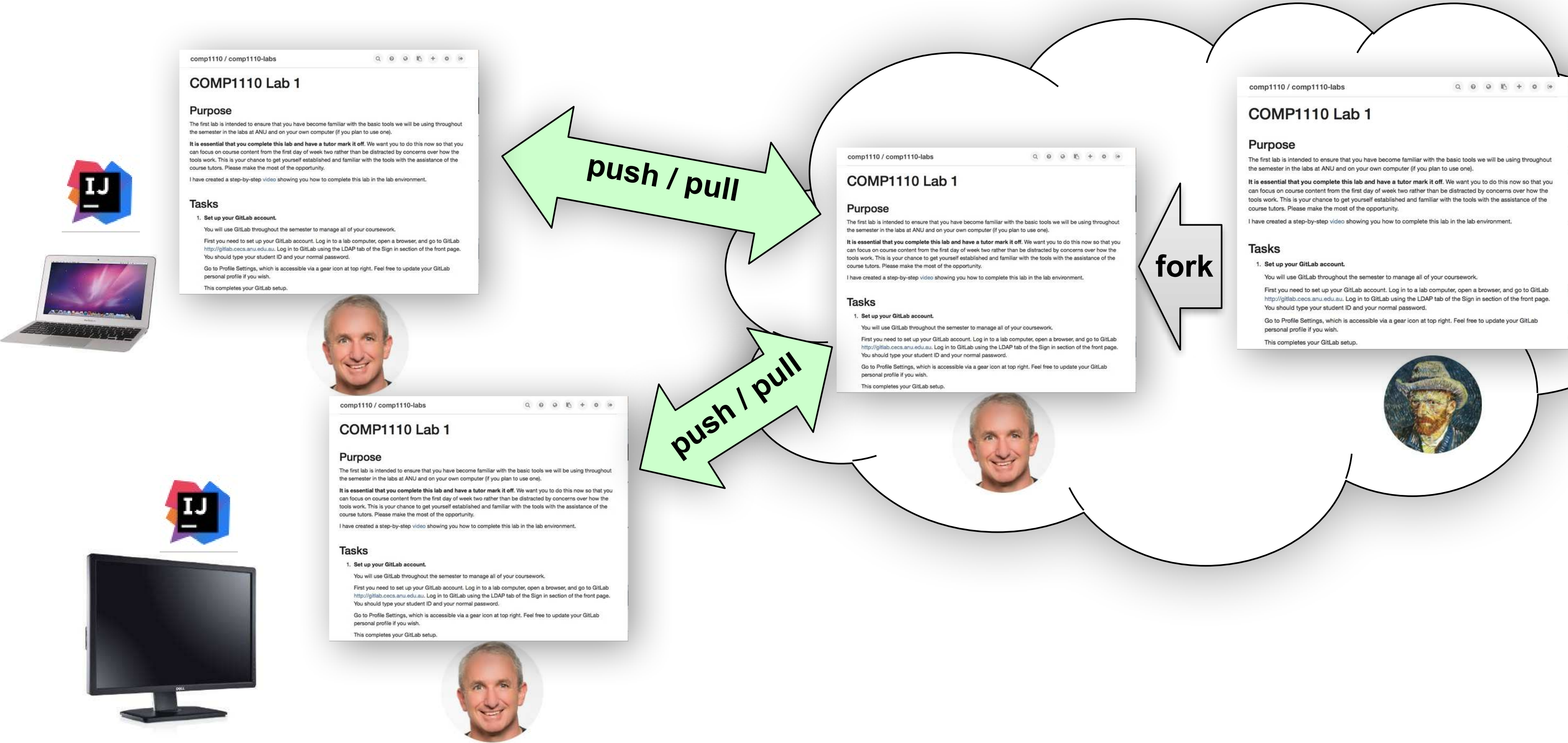
- Indispensible software engineering tool
- Solitary work
  - Personal audit trail and time machine
  - Establish when bug was introduced
  - Fearlessly explore new ideas (roll back if no good)
- Teamwork
  - Concurrently develop
  - Share work coherently

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



IntelliJ Git Integration

- Clone an existing repository:
  - “Get from VCS” on splash screen
- Other operations:
  - Git menu
  - right mouse click > Git

S02 Revision Control

Git

ANU – School of Computing – Structured Programming 1110 / 1140 / 6710

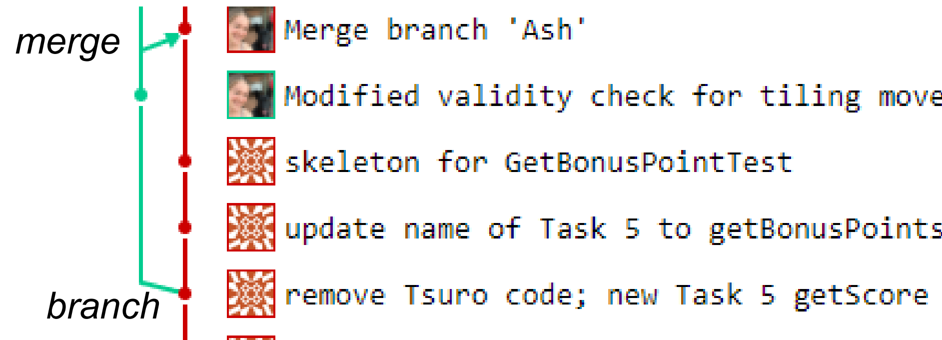
Git Concepts

- 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 (*fetch* new changes from a remote repo and merge / rebase locally)
- ✓ Update (update your working version – specific to IntelliJ)
- Merge / Rebase
- Reset and Revert

Git Commits

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

- Globally unique commit ID (large hexadecimal number)
- Parent – child relationship
  - Single parent, single child is simple case
  - Multiple children indicates a **branch**
  - Multiple parents indicates a **merge**
- 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 (Git -> Show Git Log, select the revision right click "Checkout 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, Revert and Rebase

You can reset your local state to a particular commit (throwing away un-pushed changes whether committed or not) with **reset**.

You can also **revert** any particular commit. This amounts to applying a counteracting commit.

WARNING: The following commands will cause trouble if they “modify” commits that have been previously pushed:

You can amend a commit message, add more changes with **amend**.

You can interactively remove, combine, reorder and edit commits with **rebase** *interactive*.

When All Else Fails

https://xkcd.com/1597/

S03 Software Development Teams

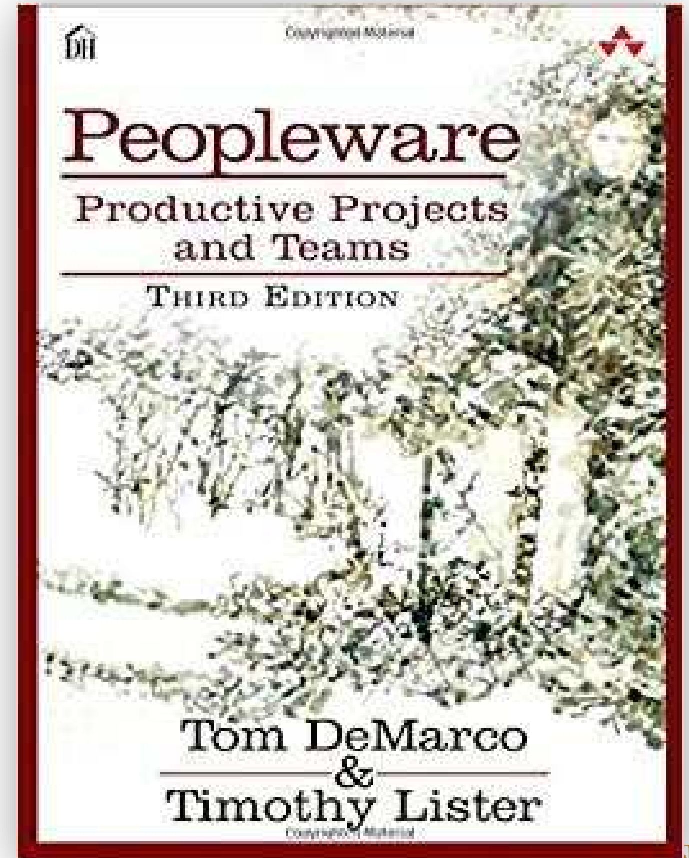
Importance of people in software engineering
Understanding team effectiveness
Conflict and conflict resolution
Code of conduct

If you find yourself concentrating on the technology rather than the sociology, you’re like the vaudeville character who loses his keys on the dark street and looks for them at the adjacent street because, as he explains, “ The light is better there”.

Tom DeMarco

Q: Why Do Software Projects Fail?

A: People



Understanding Team Effectiveness

- Major Google study of 180 teams world-wide
  - Gathered data on team members (attitudes, skills, personality, etc.)
  - Used statistics to identify factors that correlated with performance

https://rework.withgoogle.com/guides/understanding-team-effectiveness/

Understanding Team Effectiveness

- Factors:
  - Co-location of teammat[illegible]
  - Consens[illegible]iven d[illegible]n making
  - Extroversi[illegible]nembers
  - Individual pe[illegible]nce of team members
  - Workload
  - Seniori[illegible]
  - Team siz[illegible]
  - Tenure

These **did not** significantly impact the performance measure used by Google in their study.

This does not mean that these are not important factors in other settings or other regards.

1
Psychological Safety
Team members feel safe to take risks and be vulnerable in front of each other.
2
Dependability
Team members get things done on time and meet Google's high bar for excellence.
3
Structure & Clarity
Team members have clear roles, plans, and goals.
4
Meaning
Work is personally important to team members.
5
Impact
Team members think their work matters and creates change.
re:Work

Conflict Resolution Strategies

Conflict is a part of any work environment.

Working under stress is bound to cause problems.

Stephanie Ray, 2018, 10 Conflict Resolution Strategies that Actually Work

Conflict Resolution Strategies

1) Define Acceptable Behavior
2) Don’t Avoid Conflict
3) Choose a Neutral Location
4) Start with a Compliment
5) Don’t Jump to Conclusions
6) Think Opportunistically, Not Punitively
7) Offer Guidance, Not Solutions
8) Constructive Criticism
9) Don’t Intimidate
10) Act Decisively

Stephanie Ray, 2018, 10 Conflict Resolution Strategies that Actually Work

https://comp.anu.edu.au/courses/comp1110/policies/#code-of-conduct

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)

S04 Test-Driven Development

Test-driven development (TDD)
JUnit

ANU – School of Computing – Structured Programming 1110 / 1140 / 6710

Types of Tests

- **Unit tests**: testing individual “units” / “modules”
  - In OO a unit is at the level of a **method** or **class**
  - Check the “building blocks” are functioning correctly
- **Integration tests**: the integration of multiple modules
  - Expose problems with interface of modules and interactions between them
- **System tests**: end-to-end complete system
  - Checking it meets its requirements

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*

What Makes **Good** Unit Tests?

- Isolate behaviour / reduce dependencies
- Common path / usage
- Edge cases
- Touch on all branches
- Deterministic
- Limit false positives (test fails for correct code)
- Coverage

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`, `@BeforeEach`, `@BeforeClass`, `@AfterEach`, `@AfterClass`, `@Timeout`
- JUnit can check that an exception is thrown if that is expected in a certain case
  - ```
    Assertions.assertThrows(
      ArithmeticException.class,
      () -> myMethod());
    ```
    ```

S05 Code Review

Software Complexity
Code Review
Software Design
Comments and Documentation

Software Complexity

```
++++++++[>++++[>++>+++>+++>+<<<<-]>+>+>->>+
[<]<-]>>.>---.+++++++..+++.>>.<-.<.++
+.------.--------.>>+.>++.
```

- “Hello World” in the BrainF#@k language (apparently: source wikipedia)
- Syntax only 8 characters, *Turing complete*
- Simple or complex?

Software Complexity

- The International **Obfuscated** C Code Contest
- Yusuke Endoh one of the 2020 winners: Minesweeper Solver

```
                              #include/**/<time.h>
                              #include <ncurses.h>
                              # include <stdlib.h>
          /**                 */#define O()for(y-=                  !!\
         y;y<H&&           /*...Semi-Automatic.*/y<            p/W+2;\
       y++)for(x=p%   W,x-=!!/*..MineSweeper...*/x;x<W&&   x<p%W+2;x++)
     #define _(x,y)COLOR_##x,COLOR_##y /*  click / (R)estart / (Q)uit  */
   #define Y(n)attrset(COLOR_PAIR(n)),mvprintw(/* IOCCC2019 or IOCCC2020 */
  typedef int I;I*M,W,H,S,C,E,X,T,c,p,q,i,j,k;char G[]=" x",U[256];I F(I p){ I
     r=0,x,y=p/W,q;O()g=y*W+x,r+=M[g]^=p-g?(M[g]&16)<<8:0;return r;}I K(I p
       ,I f,I g){ I x=(g+     f/256)%16-(f+g/256)%16,y=p/W,c=0,n=g/4096
        ,m=x==n?0:x==g          /16%16-f/16%16-n?256:-1; if(m+1)O()if
       ((4368&M[n=y*W            +x])==4112){ M[c=1,n]=(M[n]&~16)|m; }
      return c;}void             D(){I p,k,o=0,n=C,m=0,q=0;if(LINES-1<H
     ||COLS/2<W)clear            (),Y(4)LINES/2,COLS/2-16,"Make the ter\
minal bigger!");else{for        (p=0;p<S;o+=k==3,Y(k)p/W+1,p%W*2,G),p++)G[1]=""
"_*!..12345678"[k=E?256&M[p      ]?n--,2:E-2||M[p]%2<1?M[p]&16?q=p,m++,3:4+F(p)%16:
1:3];k=T+time(0);T=o||T>=0||E-1?T:k;k=T<0?k:T;Y(7)0,0,"%03d%*s%03d",n>999?999:n,W*
2-6,"",k>999?999:k);Y(9)0,W-1,E>1?"X-(":E-1||o?":-)":"8-)");M[q]|=256*(n==m&&n); }
refresh();}short B[]={_(RED,BLACK),_(WHITE,BLUE),_(GREEN,RED),_(MAGENTA,YELLOW),_(
CYAN,RED)};I main(I A,char**V){MEVENT e;FILE*f;srand(time(0));initscr();for(start\
_color();X<12;X++){init_pair(X+1,B[X&&X<10?X-1:2],B[X?X<3?2:1:0]);}noecho();cbreak
();timeout(9);curs_set(0);keypad(stdscr,TRUE);for(mousemask(BUTTON1_CLICKED|BUTTO\
N1_RELEASED,0);;){S=A<2?f=0,W=COLS/2,H=LINES-1,C=W*H/5,0:fscanf(f=fopen(V[A-1],"r"
      ),"%d %d %d",&W,&H,&C)>3; ;S+=W*H;M=realloc(M,S*sizeof(I)*2);for(i=0
       ;i<S;i++)!f?M[i]=i,i&&(k=M[j=rand()%i],M[j]=M[i],M[i]=k):fscanf(f,
        "%d",M+i);if(f)fclose(f);T=E=X=0;for(clear();D(),c=getch(),c-'r'
         &&(c-KEY_RESIZE||E);){ if(c=='q'){ return(endwin(),0); }if(c==
       KEY_MOUSE&&getmouse(&e)==OK&&e.x/2<W&&e.y<=H){if(!e.y&&(W-2<e.x&&
     e.x<W+2)){break;}p=e.x/2+e.y*W-W;if(p>=0){if(!E){for(i=0;i<S;i++)M[S+M
   [i]]=i,M[i]=16+(M[i]<C);C-=M[p]&1;M[p]=16;E=1;T=-time(0);}if(E<2)M[p]&=(M[p]
    &257)==1?T+=time(0),E=2,273:257;}}for(p=0;p<S&&E==1;M[p++]&=273){}for(i=
      (X+S-1)%S;E==1&&i!=X;X=(X+1)%S){if(!(M[p=M[X+S]]&272)){if(K(p,c=F(p)
       ,0)){goto N;}   for(k=p/W-2,k=k<0?0:k;k<p/W+3&&k   <H;k++)for(j=
         p%W-2,j           =j<0?0:j;j<W&&j<p%W+3;)if             (!(M[q=
          k*W                +j++]&272)){ if(K(p,                c,F
                              (q))){ goto N; }F(q)
                              ; }F(p); }}N:; } } }
                              /*(c)Yusukse Endoh*/
```

What is Software Complexity?

- ***Accidental* Complexity**
  - Software that is designed or presented in a way that is more difficult for a **human to understand, use and modify** *than it needs to be*.
  - It is difficult to write elegant, clear, reusable code.
- ***Essential* Complexity**
  - Inherent to the problem being solved. Irreducible.
- **Not to be confused with** computational complexity (about performance).

Software Complexity

- Some **contributing factors**:
  - Poorly named variables
  - Not following conventions / inconsistency
  - Interlinking many components
  - Unstated assumptions
  - Non-local changes, unintuitive side-effects
  - Duplication / lack of encapsulation / exposure to details
- Often **incrementally** works its way into a project, e.g., *feature creep*, dealing with *legacy*.

Code Review

- One or more people review code who are removed from the implementation.
- Commonly done for a specific change (e.g., set of git commits) but can also be done for a complete project / implementation.
  - **Fix** a specific bug
  - **Implement** a new feature
  - **Refactor** part of the code
- Gitlab offers a “merge request” workflow (“pull request” on github) where reviewers / maintainers review the changes **before** they are merged into the mainline branch.

Code Review Motivations

- Barrier to ensure project remains **maintainable**.
  - Improve implementation / quality.
  - Clarify code, double-check edge cases.
  - On-balance rejection of a feature (accidental or essential complexity).
- Second pair of eyes: potentially less biased, can consider bigger picture, can bring new insight.
- Effective way to **learn** a new code-base and a team's processes / conventions. Highlights interrelated parts.
- Can catch some bugs before reaching production… but implementer really should have adequate tests developed and passing.

Doing a Code Review

- **Objective**: is it in scope of this project
- **Functionality** (for end-users and developers):
  - does it do what is intended
  - edge cases / bugs
  - might have to run code for UI changes etc
- **Tests**: present, appropriate
- **Complexity**: design minimises / encapsulates complexity
- **Good names**: convey information and not too long
- **Comments**: help to understand decisions and the why, not repeating code, appropriately documenting interfaces
- **Conformance** to project style guide / conventions.

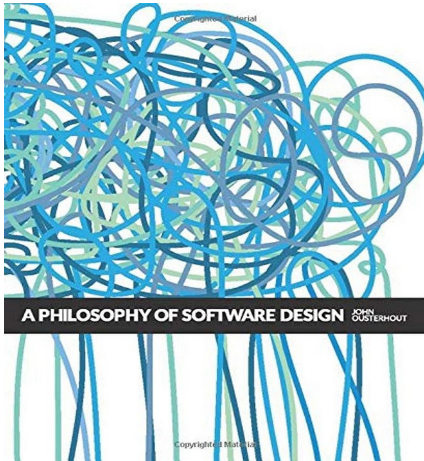
Further Tips

- Be considerate.
- Point out things that are good!
- Clearly label *nitpicks* as such.
- No code is ever perfect. Tailor to circumstances:
  - flight control software
  - a game

Good Software Design

- Many opinions. Conventions / preferences vary between communities.
- Recommendation:

  A Philosophy of Software Design, John Ousterhout



- Design principles
- Red flags

Some Principles (Ousterhout)

- **Deep “modules”** (method, class, package, or module)
  - Simple interfaces* (narrow)
  - Encapsulate lots of complexity (depth)
  - General-purpose
- Prefer **simple interface** over simple implementation
- Design **errors out of existence**
- Design for **ease of reading**, not ease of writing
- Extra: Don’t Repeat Yourself (**DRY**) and **SOLID** principles

* Interfaces in the broad sense, not just the Java keyword

Some Red Flags (Ousterhout)

- **Shallow module**: interface not much simpler than implementation
- **Overexposure**: user needs to be aware of rarely-used features
- **Repetition**: non-trivial code is repeated
- **Conjoined methods**: methods are so co-dependent that you have to understand implementation of both
- **Comment repeats code**
- **Hard to name entity**
- Extra: **Deeply nested control-flow blocks**

Code Comments / Documentation

- **Class or method comments – always for** `public`
  - How to use, edge cases, side-effects, pre/post-conditions, invariants, explain abstraction, examples.
  - Should not leak the implementation details.
- **Implementation comments – as required**
  - Give intuition where implementation is non-obvious to a likely contributor / your future self
  - Highlight where edge cases are handled if hidden
  - Rationale for the design if not the obvious choice
  - Should not just repeat code