Software Engineering Project

Morgan Ericsson

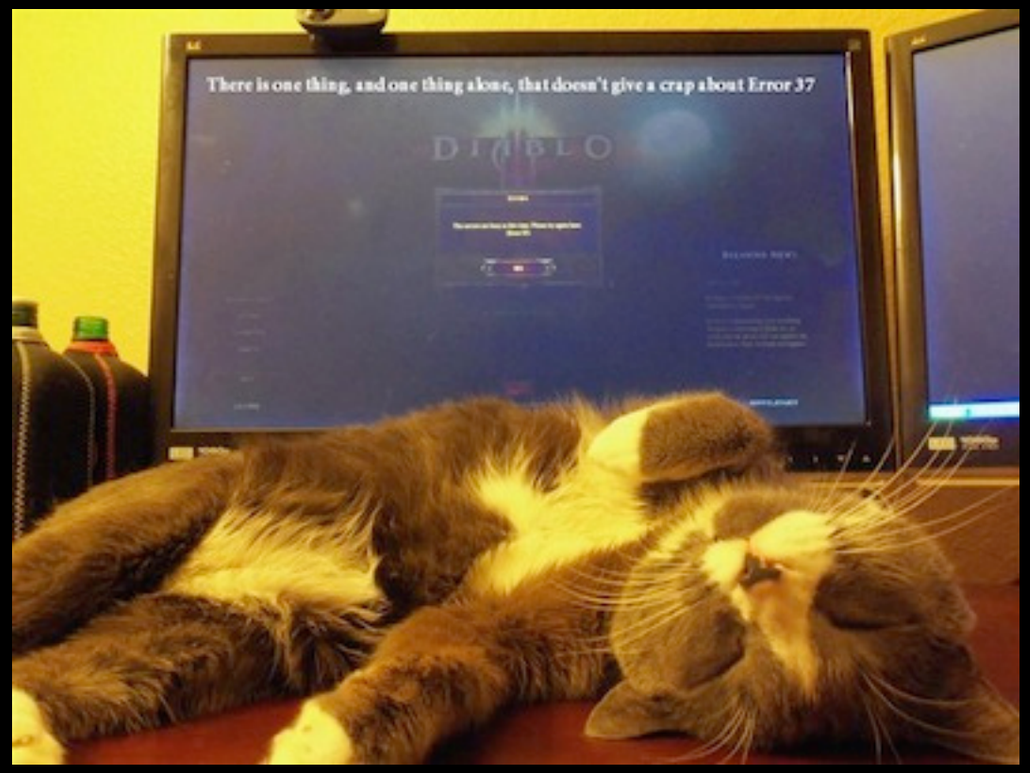
morgan@cse.gu.se

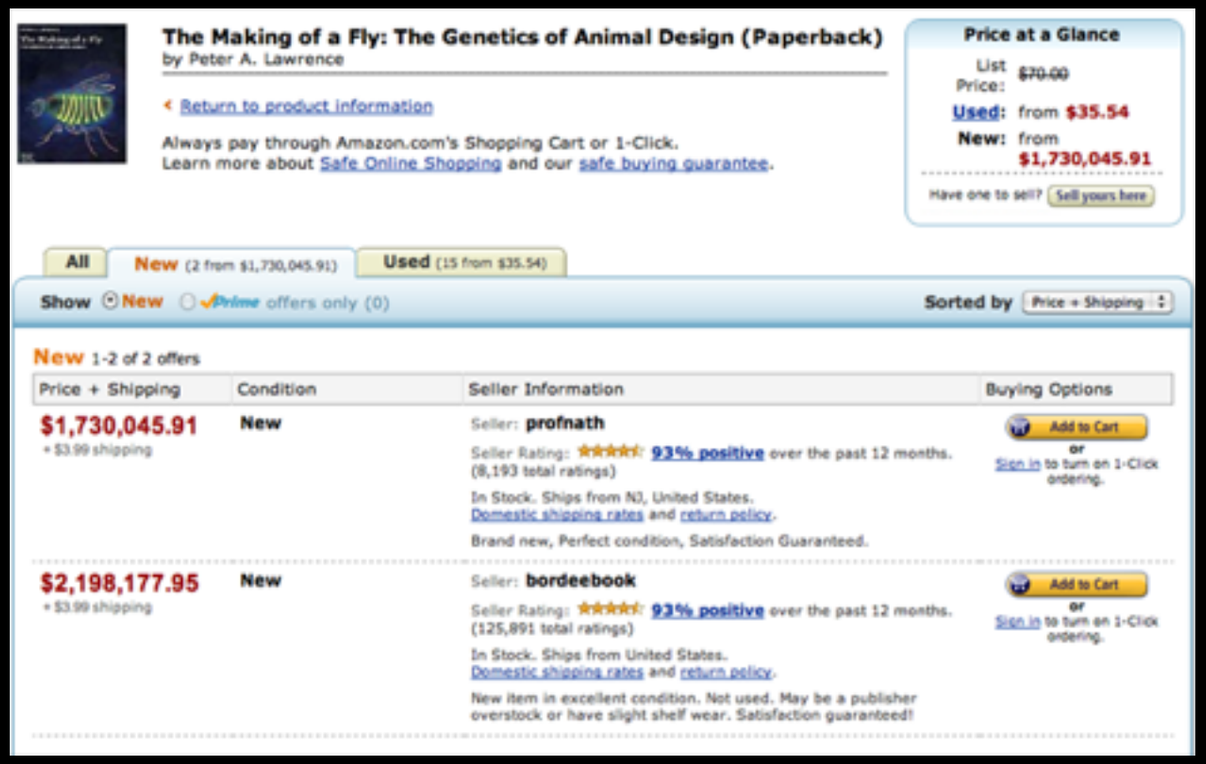
@morganericsson

morganericsson

UNIVERSITY OF GOTHENBURG

CHALMERS





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

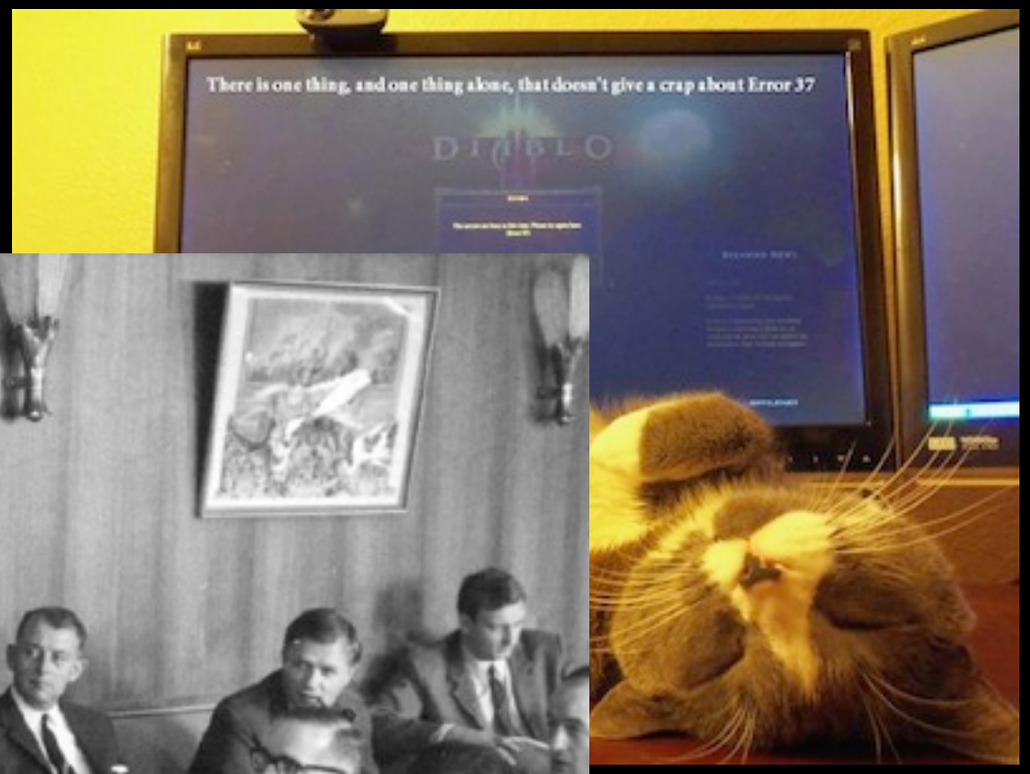
New 1-2 of 2 offers

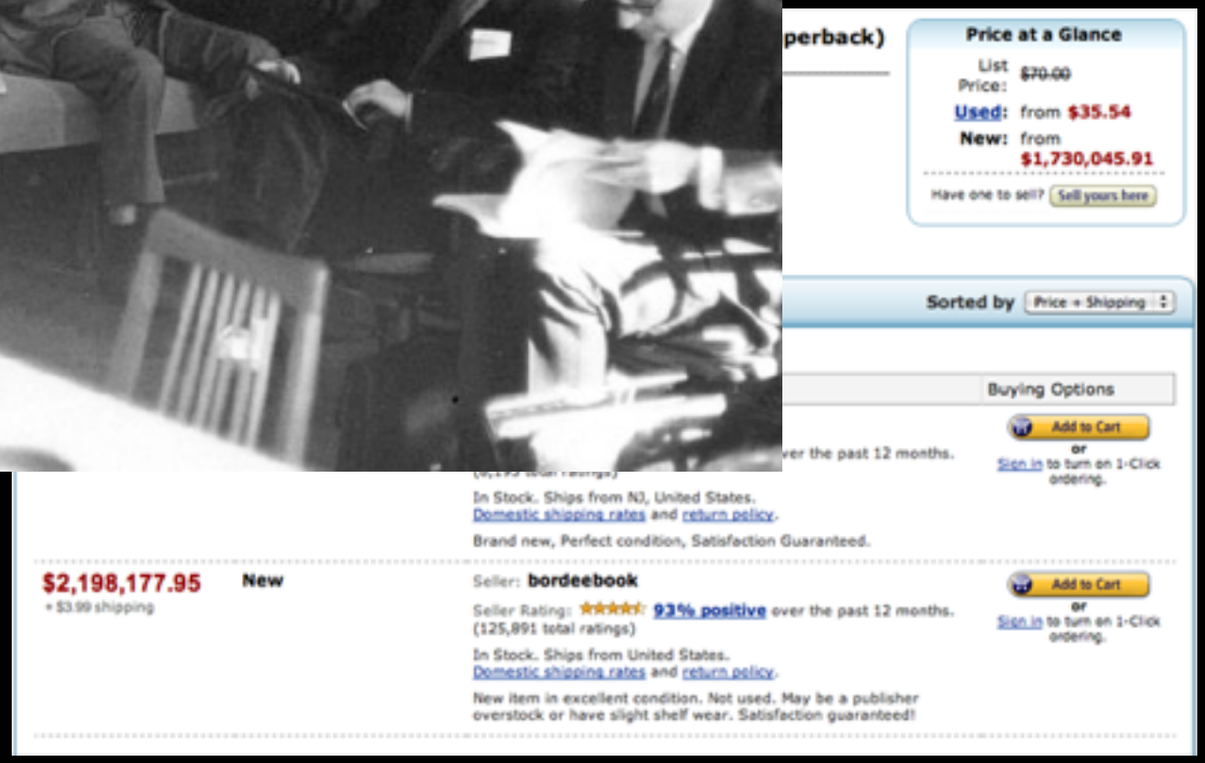
Price + Shipping	Condition	Seller Information	Buying Options
$1,730,045.91 + $3.99 shipping	New	Seller: profnath Seller Rating: 93% positive over the past 12 months. (8,193 total ratings) In Stock. Ships from NJ, United States. Domestic shipping rates and return policy. Brand new, Perfect condition, Satisfaction Guaranteed.	Add to Cart or Sign in to turn on 1-Click ordering.
$2,198,177.95 + $3.99 shipping	New	Seller: bordeebook Seller Rating: 93% positive over the past 12 months. (125,891 total ratings) In Stock. Ships from United States. Domestic shipping rates and return policy. New item in excellent condition. Not used. May be a publisher overstock or have slight shelf wear. Satisfaction guaranteed!	Add to Cart or 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"





Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968

Five Steps

Requirements

Design

Implementation

Quality Assurance

Evolution

A sequential model of software development

Change is Ubiquitous

Design

Requirements

Implementation

Quality Assurance

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 and typically pre-study heavy
- Assumes (more) static environment
- Longer iterations
- Change management intensive
- 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-organising teams
- Continuous “customer” interaction
- **NOT UNPLANNED, rather adaptive!!!**

In reality/practice

- Many different
  - processes
  - methodologies
  - practices
  - …

eXtreme Programming

- XP is what it says, an extreme way of developing software
  - if a practice is good, then do it all the time
  - if a practice causes problems with project agility, then do not do it

eXtreme Programming

- Team, 3-10 programmers + 1 customer
- Iteration, tested and directly useful code
- Requirements, user story, written on index cards
- Estimate development time per story, prio on value
- Dev starts with discussion with expert user

eXtreme Programming

- Programmers work in pairs
- Unit tests passes at each check-in
- Stand-up meeting daily: Done? Planned? Hinders?
- Iteration review: Well? Improve? => Wall list

XP practices

- Whole Team (Customer Team Member, on-site customer)
- Small releases (Short cycles)
- Continuous Integration
- Test-Driven development (Testing)
- Customer tests (Acceptance Tests, Testing)
- Pair Programming

XP practices

- Collective Code Ownership
- Coding standards
- Sustainable Pace (40-hour week)
- The Planning Game
- Simple Design
- Design Improvement (Refactoring)
- Metaphor

Whole Team

- Everybody involved in the project works together as ONE team.
- Everybody on the team works in the same room (open workspace)
- One member of this team is the customer, or the customer representative

Small Releases

- The software is frequently released and deployed to the customer
- The time for each release is planned ahead and are never allowed to slip. The functionality delivered with the release can however be changed right up to the end
- A typical XP project has a new release every 3 months
- Each release is then divided into 1-2 week iterations

Continuous Integration

- Daily build
  - A working new version of the complete software is released internally every night
- Continuous build
  - A new version of the complete software is built as soon as some functionality is added, removed or modified

Test-Driven Development

- No single line of code is ever written, without first writing a test that tests it
- All tests are written in a test framework such as JUnit so they become fully automated

Customer Tests

- The customer (or the one representing the customer) writes tests that verifies that the program fulfils his/her needs

Pair Programming

- All program code is written by two programmers working together; a programming pair
- Working in this manner can have a number of positive effects:
  - better code quality
  - fun way of working
  - skills spreading
  - ...

Collective Code Ownership

- All programmers are responsible for all code
- You can change any code you like, and the minute you check in your code somebody else can change it
- You should not take pride in and responsibility for the quality of the code you written yourself but rather for the complete program

Coding standards

- In order to have a code base that is readable and understandable by everybody the team should use the same coding style

Sustainable Pace

- Work pace should be constant throughout the project and at such a level that people do not drain their energy reserves
- Overtime is not allowed two weeks in a row

Simple design

- Never have a more complex design than is needed for the current state of the implementation
- Make design decisions when you have to, not up front

Design Improvement

- Always try to find ways of improving the design
- Since design is not made up front it needs constant attention in order to not end up with a program looking like a snake pit
- Strive for minimal, simple, comprehensive code

Metaphor

- Try to find one or a few metaphors for your application
- The metaphors should aid in communicating design decisions and intends
- The most well known software metaphor is the desktop metaphor

User Stories

- One or more sentences in the everyday or business language
- Captures what a user does or needs to do as part of his or her job function
- Quick way of handling requirements without formalised requirement documents
- Respond faster to rapidly changing real-world requirements

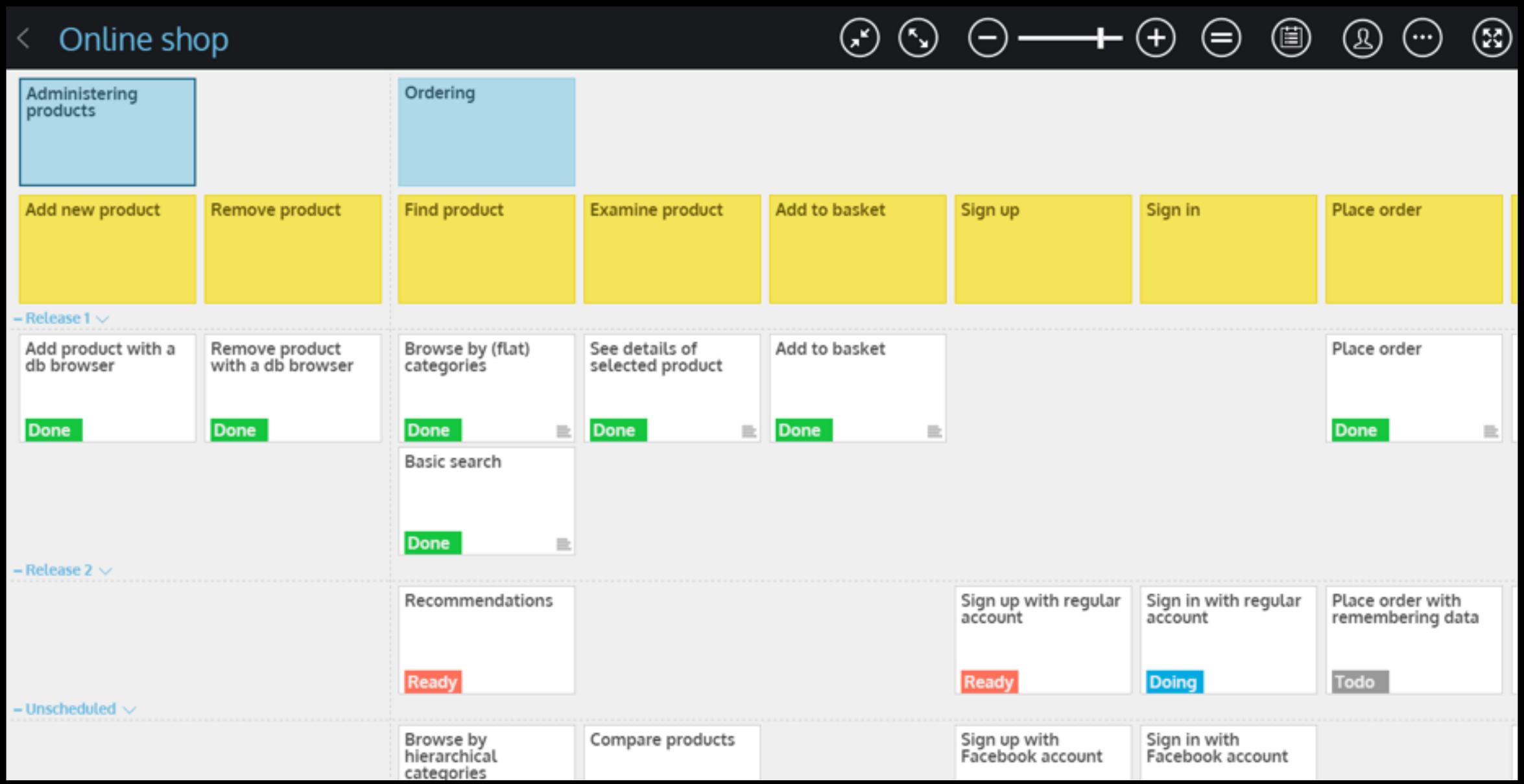
User Stories

- "As a <role>, I want <goal/desire> (so that <benefit>")
- "As <who> <when> <where>, I <what> because <why>."
- *"As a user, I want to search for my customers by their first and last names."*
- *"As a user closing the application, I want to be prompted to save if I have made any change in my data since the last save."*

Benefits

- Represent small chunks of business value that can be implemented in a period of days to weeks
- Needs very little maintenance
- Allows the projects to be broken into small increments
- Are suited to projects where the requirements are volatile or poorly understood. Iterations of discovery drive the refinement process
- Makes it easier to estimate development effort
- Requires close customer contact throughout the project so that the most valued parts of the software get implemented

Story Maps


Online shop
Administering products
Ordering
Add new product
Remove product
Find product
Examine product
Add to basket
Sign up
Sign in
Place order
Release 1
Add product with a db browser
Done
Remove product with a db browser
Done
Browse by (flat) categories
Done
See details of selected product
Done
Add to basket
Done
Place order
Done
Basic search
Done
Release 2
Recommendations
Ready
Sign up with regular account
Ready
Sign in with regular account
Doing
Place order with remembering data
Todo
Unscheduled
Browse by hierarchical categories
Compare products
Sign up with Facebook account
Sign in with Facebook account

Example

Roles

- **Instructor**: Expected to use the website frequently, often once a week. Through the company's telephone sales group, an Instructor frequently places similar orders (for example, 20 copies of the same book). Proficient with the website but usually somewhat nervous around computers. Interested in getting the best prices. Not interested in reviews or other "frills."
- **Experienced Sailor**: Proficient with computers. Expected to order once or twice per quarter, perhaps more often during the summer. Knowledgeable about sailing but usually only for local regions. Very interested in what other sailors say are the best products and the best places to sail.

Stories for Experienced Sailors

- A user can search for books by author, title or ISBN number
- A user can view detailed information on a book. For example, number of pages, publication date and a brief description
- A user can put books into a “shopping cart” and buy them when she is done shopping
- A user can remove books from her cart before completing an order

Stories for Experienced Sailors

- To buy a book the user enters her billing address, the shipping address and credit card information
- A user can rate and review books
- A user can establish an account that remembers shipping and billing information
- A user can edit her account information (credit card, shipping address, billing address and so on)
- A user can put books into a "wish list" that is visible to other site visitors

Stories for Instructors

- A user can view a history of all of his past orders.
- A user can easily re-purchase items when viewing past orders.
- The site always tells a shopper what the last 3 (?) items she viewed are and provides links back to them. (This works even between sessions.)

Assessment

- “A user can search for books by author, title or ISBN number”
- What does that mean? Either or, or any possible combination?
- Fix!
- “A user can do a basic simple search that searches for a word or phrase in both the author and title fields”
- “A user can search for books by entering values in any combination of author, title and ISBN”

Git

What is a Version Control System (VCS)?

- Version control (source/revision) is about managing changes to documents (source code files)
- Logical way to organise and control versions
- VCS is an application (or part of) to manage the version control
  - Git, SVN, ...
  - Google Drive, Dropbox, Wikis

Why Use Version Control?

- Software exist in multiple versions
  - also deployed
- To maintain, important to access the right version
- Also, parallel development
  - features and/or fixes

Evolution of VCS

- Stage one, manual VCS
  - basically keep multiple copies of the source code (version)
  - distribute source files between developers, e.g., mail, floppies, etc.
  - track everything manually

Evolution of VCS

- Stage two, local VCS
  - SCCS and RCS
  - local file systems, and local locking

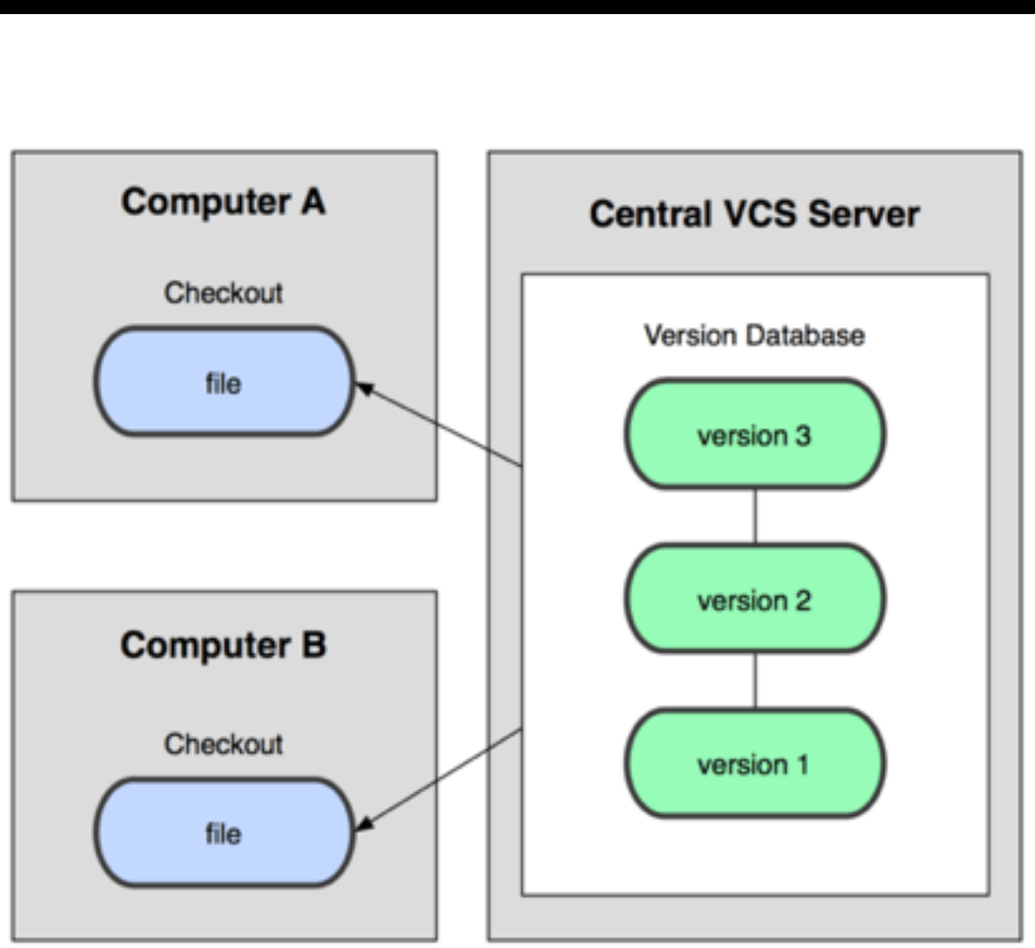
Evolution of VCS

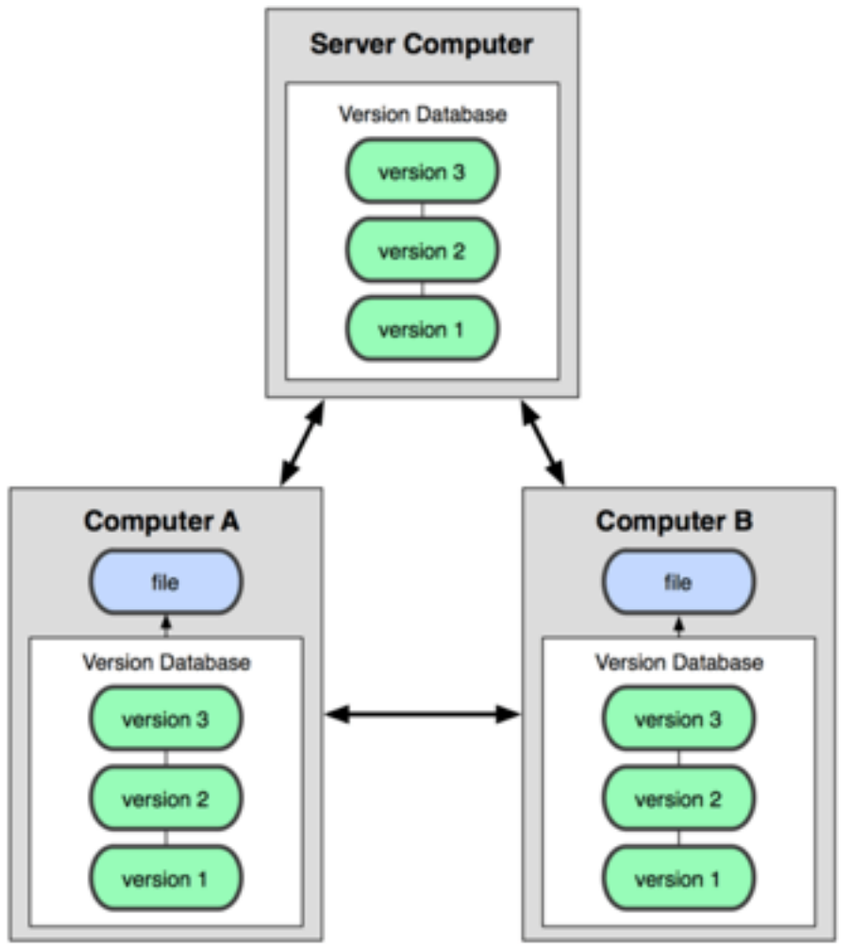
- Stage three, centralised VCS
  - CVS and SVN
    - all operations happen against a central server
    - local copy and remote repository

Evolution of VCS

- Stage four, decentralised VCS
  - Git, Mercurial, Bazaar
    - multiple "copies" of the repository that are kept in sync
    - "everything" is available locally

Evolution of VCS





Important Concepts

- Repository / Working Copy
- Change (List)
- Commit
- Trunk / HEAD
- Conflict / Merge
- History

Git

- Distributed version control
- Developed by Linus for Linux development

- Distributed
- Strong support for non-linear development
- Efficient for large projects

Getting Started

`git clone <url | path> [dir]`

- Clones a repository (more or less the entire project history)
- One of two ways to start using Git
- Locally (file system), or remotely (HTTPS, SSH, Git)

Git Basics

- Inspecting
  - status, log diff
- Modifying
  - add, rm, mv, commit
- Use `git help [<cmd>]` to get help!

Configuring Git

```
git config [--global] <key> <value>
```

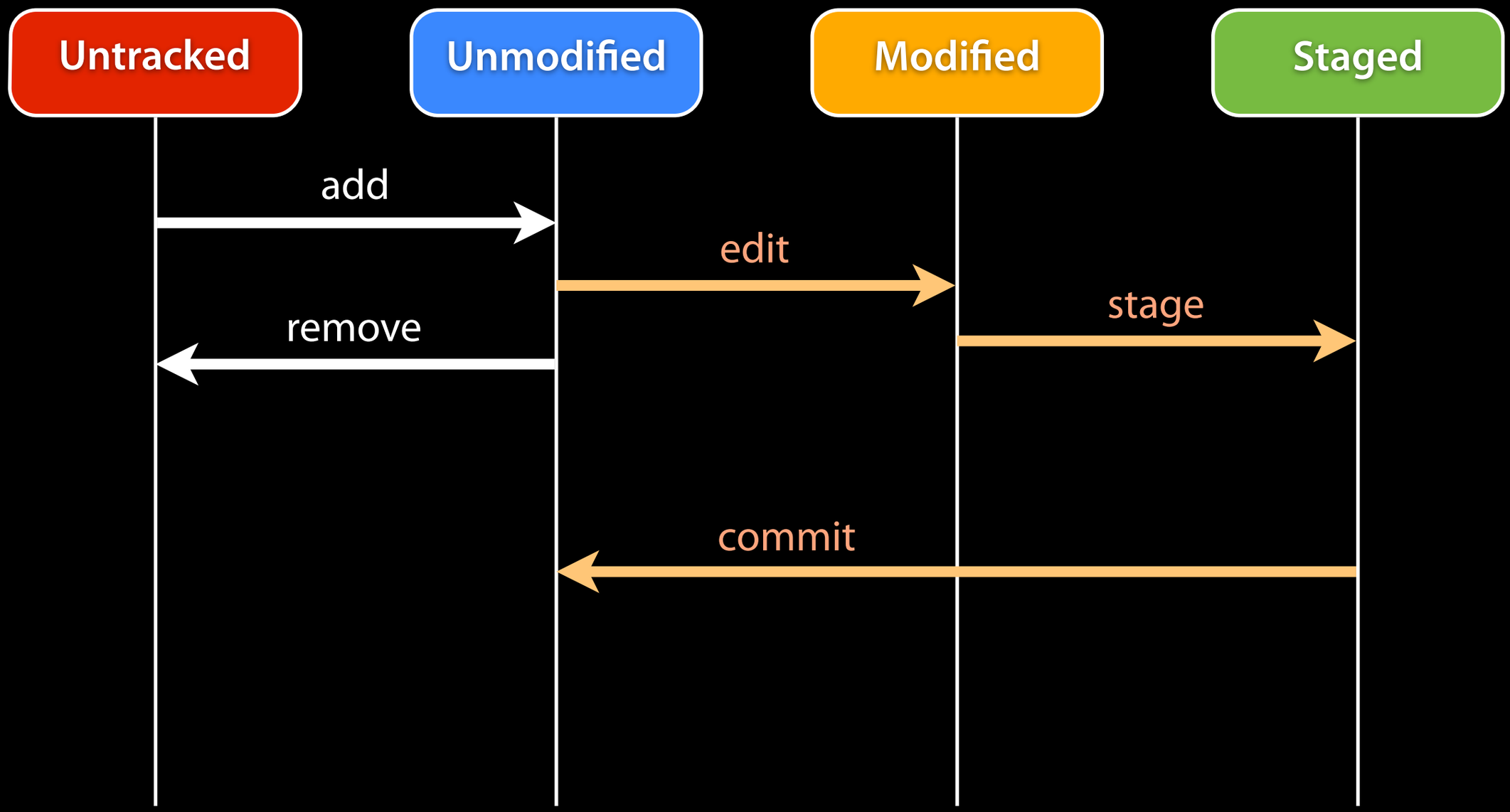
- Allows you to change settings, such as identity, editor, etc.
- Use `--global` to set information across all your repositories

Demo

File States

- **Untracked**: not managed by Git
- **Unmodified**: tracked, but not changed
- **Modified**: tracked and changed since last commit
- **Staged**: will be saved at next commit

Work Flow / Concepts



Add to the Repo

- Use `git add` to add files to the staging area (index)
- A staged file will be "saved" the next time you commit (basically create a snapshot of your repo)
- You can add files or directories, several at a time (and as many times as you like)

Committing Your Changes

- Saves a snapshot of your current repository
- Staged files become tracked by Git and are now considered "unmodified"
- When you commit, Git will ask you to comment on your changes

Removing and Moving Files

- You can also remove files ...
  - `git rm`
- ... and move files
  - `git mv`
- Note that when you remove a file, you will remove the local copy (as well)

Logging and Diffing

- You can use `git log` to show the commit history
  - so, good commit messages are important!
- And, you can use `git diff` to see changes between current and commit or local and remote

Ignoring Files

- The `.gitignore` file allows you to specify files that should never be tracked by Git
  - such as `.class` files or `.DS_Store`
  - Text file that lives in your repository
    - with a list of patterns of files and directories to ignore
  - Github can help you generate sane defaults!

Branches

- A branch is a copy of the project state at a specific time
  - a fork
- There is a main branch, master (or trunk)
  - and as many branches as you create

Why Branch

- Feature development
  - maintain a stable version while you work on something experimental
- Bug fixes
- Release management
  - the DAT255 repo has a VT2013 branch

Branching

- Use `git branch` to manage branches
  - `git branch <name>`
- Use `git checkout` to switch branches
- Other commands work as expected with branches

Merging

- When we want to integrate or incorporate changes across branches, we merge
  - `git merge <branch_to_merge>`
- There can be conflicts
  - that you might have to resolve manually

Playing With Others

- Use `git clone` to get started
- To update to or from a remote repository
  - `git fetch`, `git push` and `git pull`
  - note, pull fetches and merges
- Use `git remote` to check / manage remote repositories

Branches

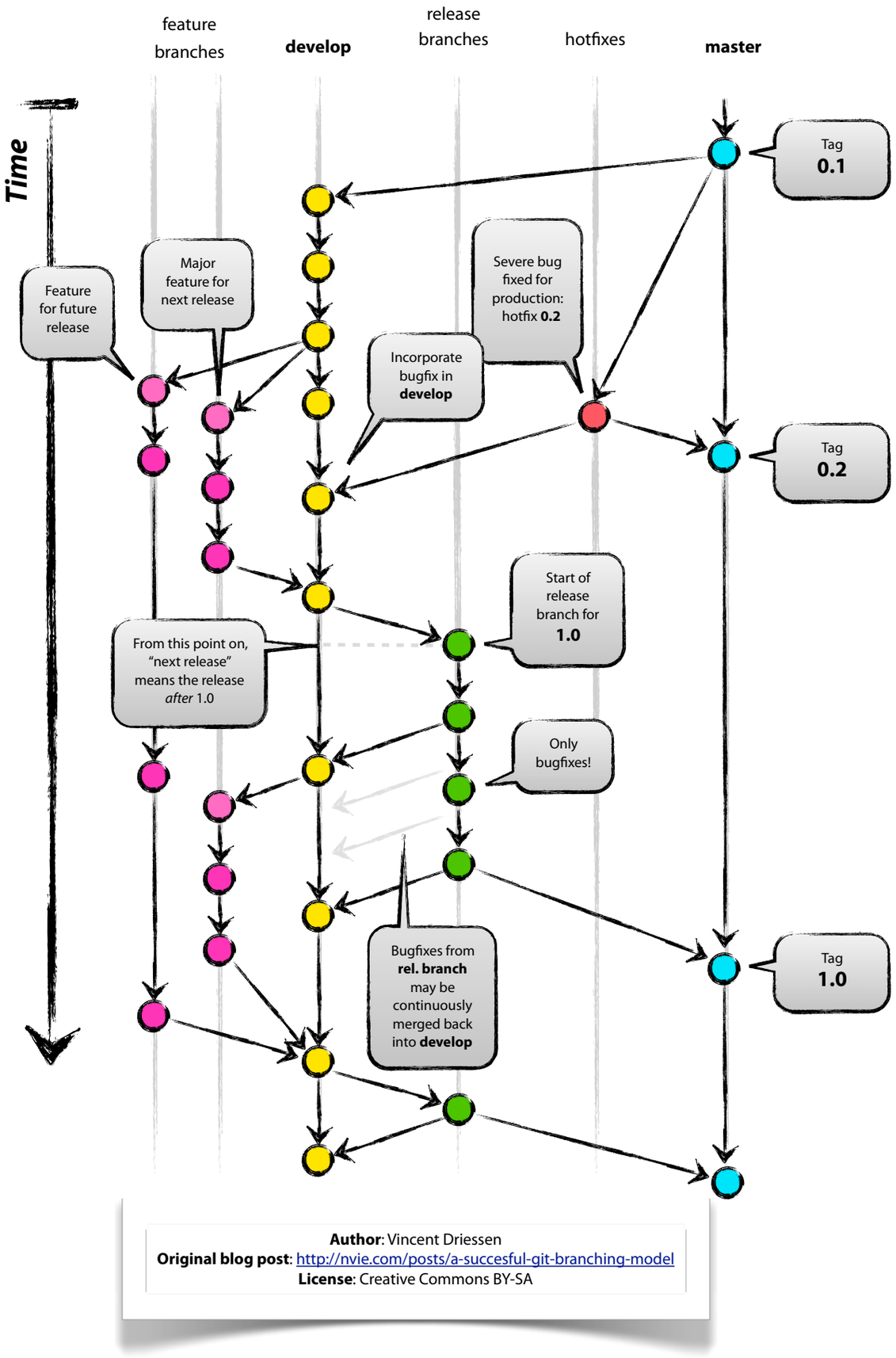
- Branches makes it more complicated, and working with other developers usually means more branches
- Remote branches are local but cannot be modified
  - you need to create a local branch to change / modify it

Releasing (and Tagging)

- A release often has a name or a version attached to it
- Use tags to mark specific commits
  - light-weight and annotated
  - `git tag [-a] <tagname> <commit>`
- Use annotated tags for releases
- Tags are not pushed by default (use `--tags`)

Best Practice

- The VCS can be a powerful tool or just another obligation (in this course)
- To make it powerful
  - set it up properly
  - find a good structure
  - embarace branching



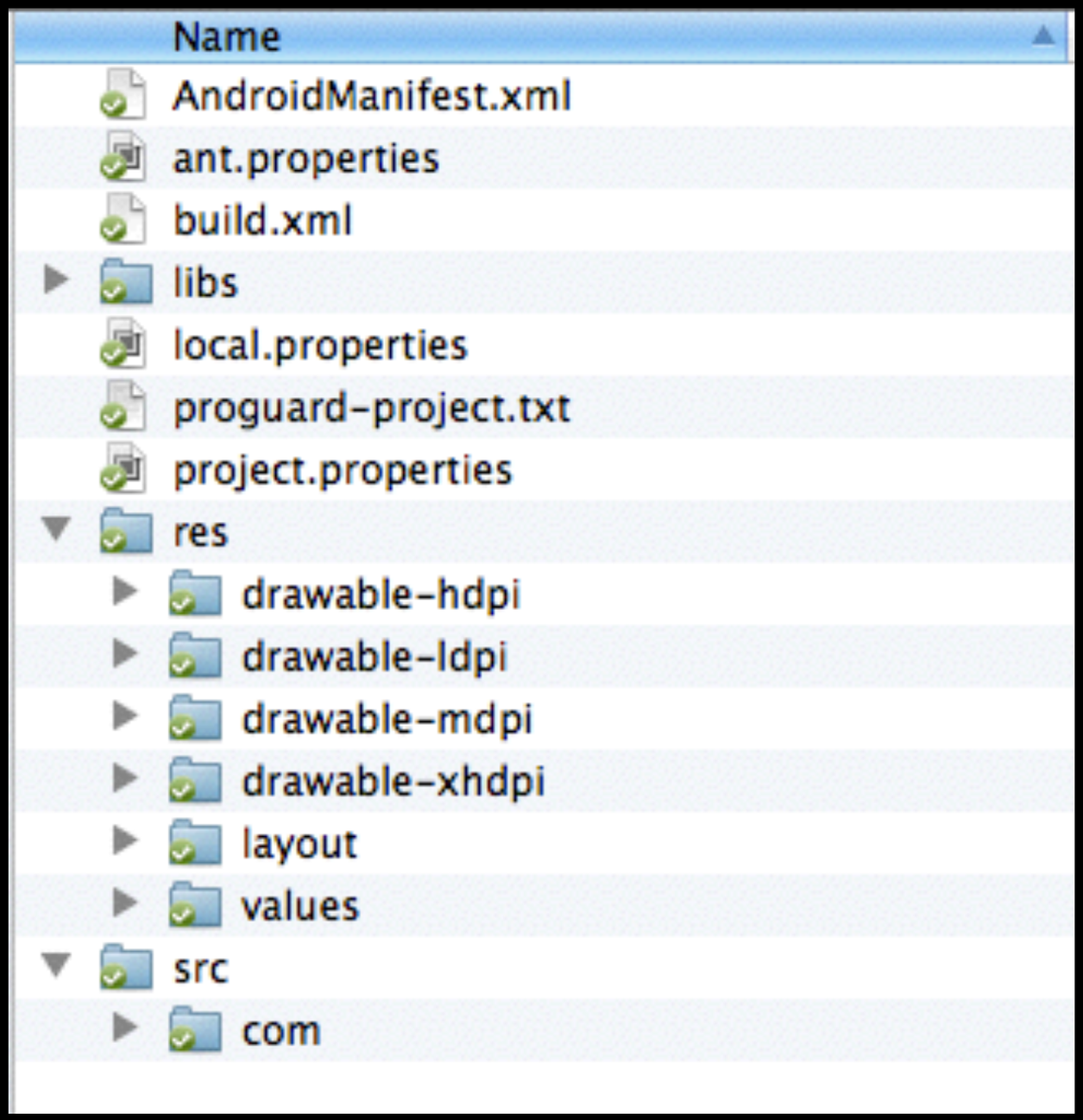
- Do not use *master* for everything!
- Use development, feature, fixes and release branches
- Tag the various releases
- Keep the branches as part of the history/log!

Demo

Android SDK

- From our perspective
- Java-based API
- XML-based layout
- tools to build/install/simulate

Structure of a project



First example

```java
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

First Example

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
     android:orientation="vertical"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     >
<TextView
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="Hello World, MainActivity"
     />
</LinearLayout>
```

Add User Input

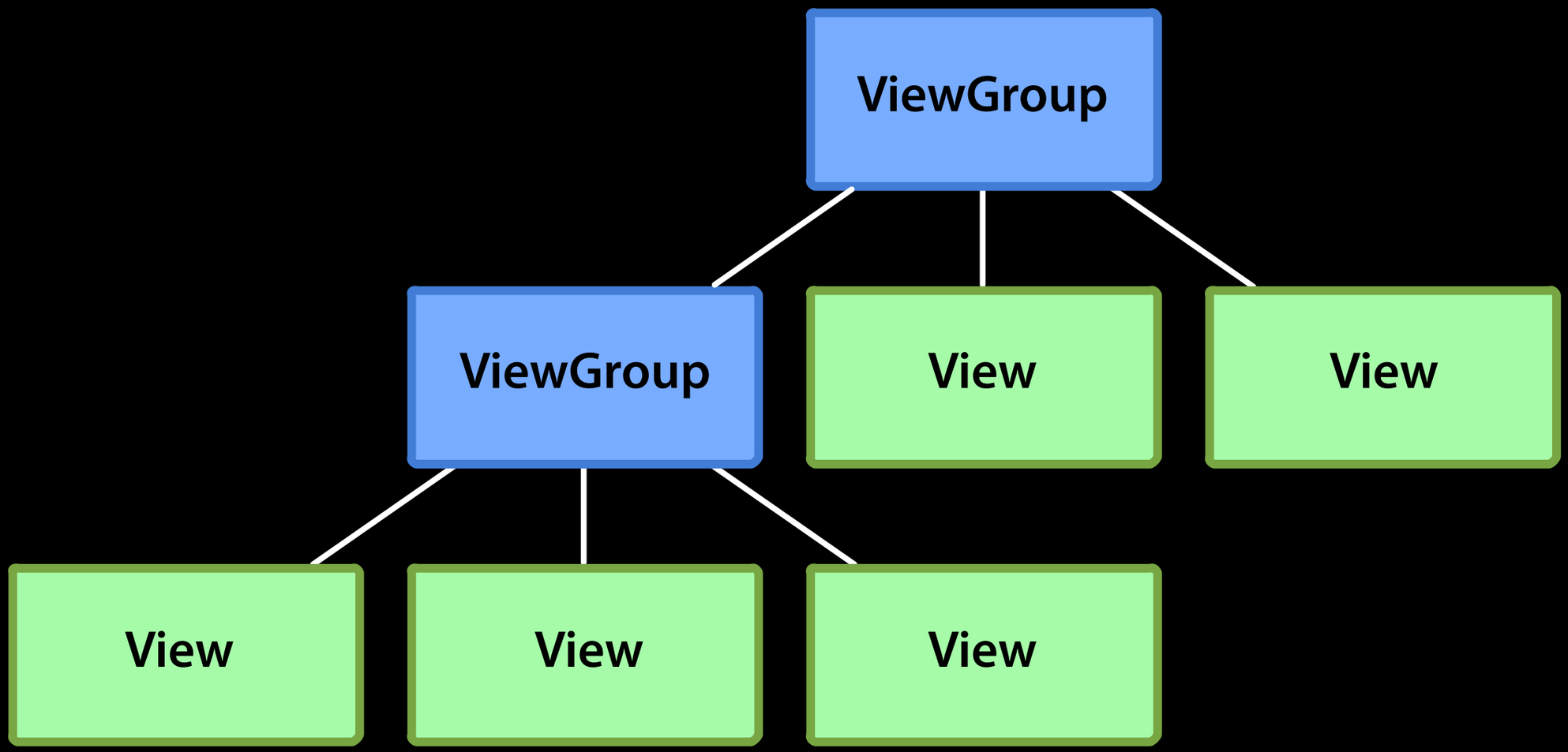
```
package com.example.helloname;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.EditText;

public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void sendName(View view) {
      TextView dn = (TextView)findViewById(R.id.displayName);
      EditText et = (EditText)findViewById(R.id.enteredName);

      dn.setText("Hello “+ et.getText().toString());
    }
}
```

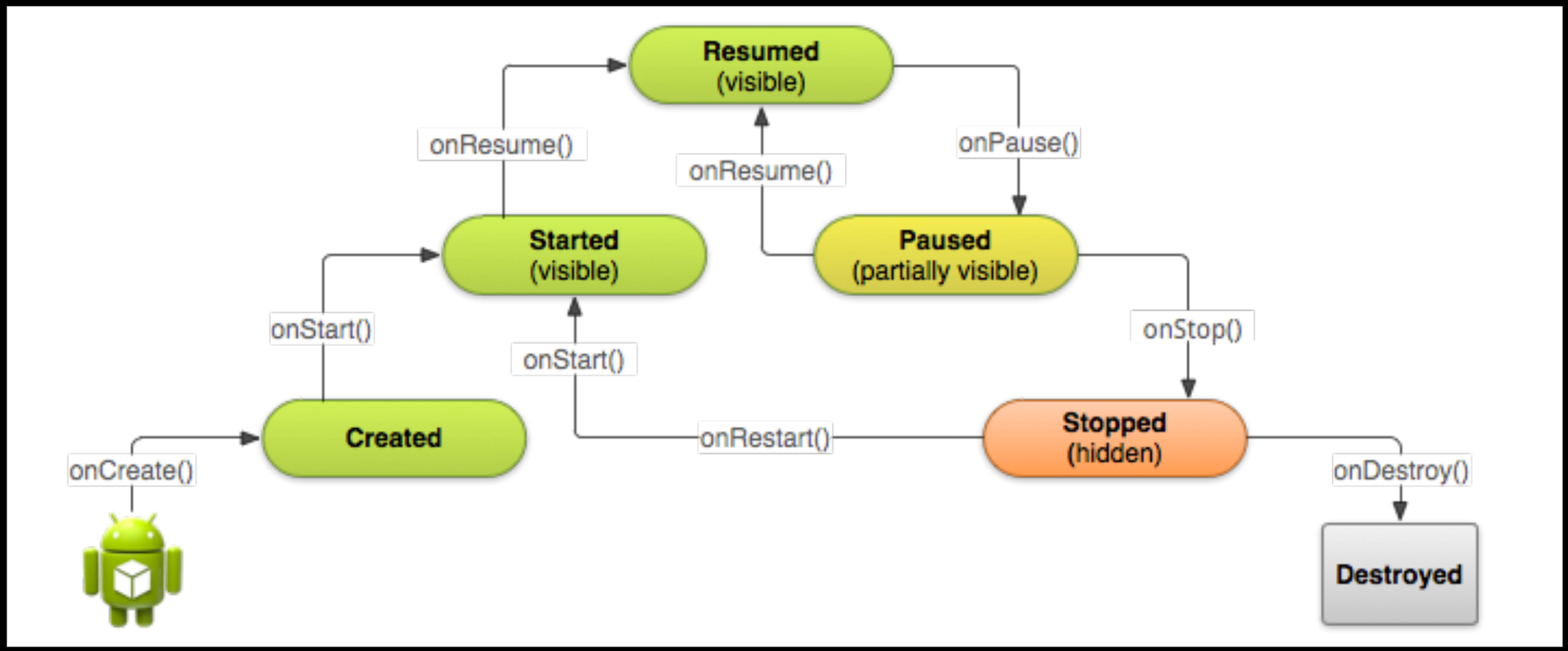
UI basics : Views



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >
<EditText android:id="@+id/enteredName"
    android:layout_width="0dp"
    android:layout_weight="1"
    android:layout_height=“wrap_content"
    android:hint="Enter your name..."
    />
```

```
<Button
    android:text = "Submit..."
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="sendName"
    />
</LinearLayout>
<RelativeLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:id="@+id/displayName"
    android:layout_centerInParent="true"
    android:textSize="24sp"
    android:textColor="#FF0000"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World, MainActivity"
    />
</RelativeLayout>
</LinearLayout>
```

Activity life cycle



Depending on the complexity of your activity, you probably don't need to implement all the lifecycle methods.

More information

- https://github.com/morganericsson/DAT255Demo
- https://github.com/morganericsson/AndroidExamples
- Video lecture (from VT2013)