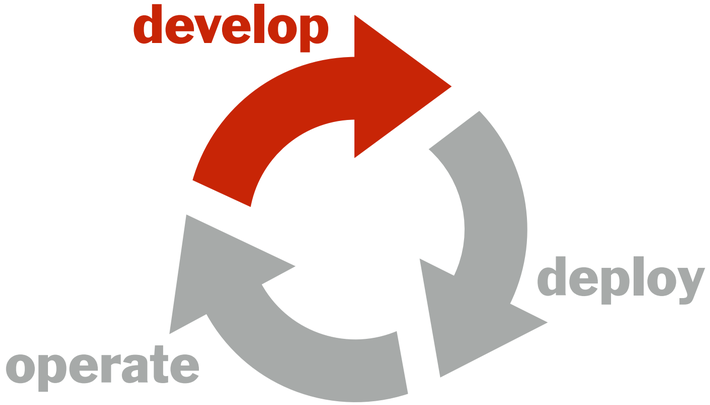
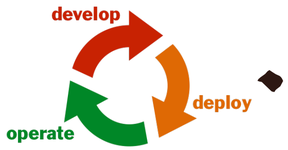
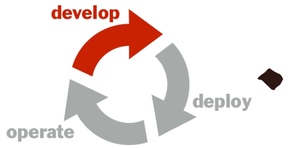
development methodologies & tools

learning objectives





- learn about software development methodologies



- learn about tools supporting those methodologies

development methodologies

waterfall

vs.

iterative

the waterfall approach

analyze

design

develop

test

deploy

measure twice
cut once

the complete value
delivered here

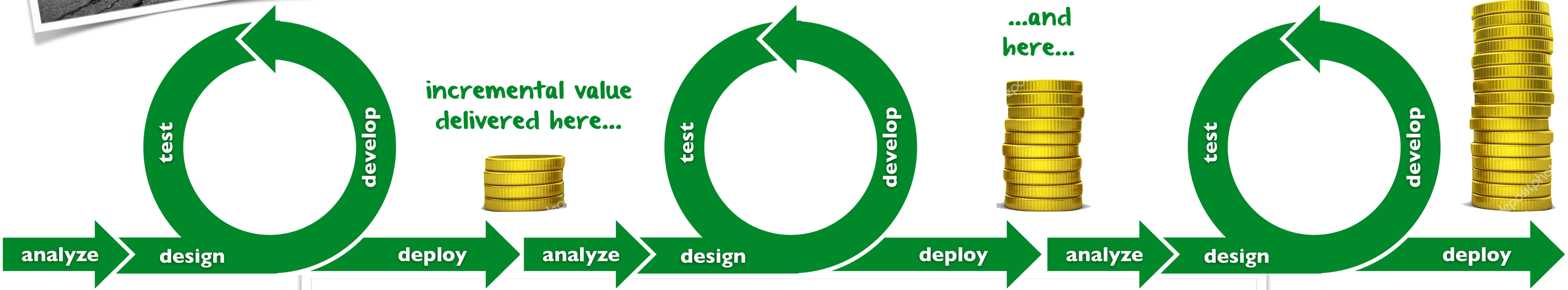
the waterfall approach

analyze → design → develop → test → deploy

value delivered here

measure twice cut once

the iterative approach

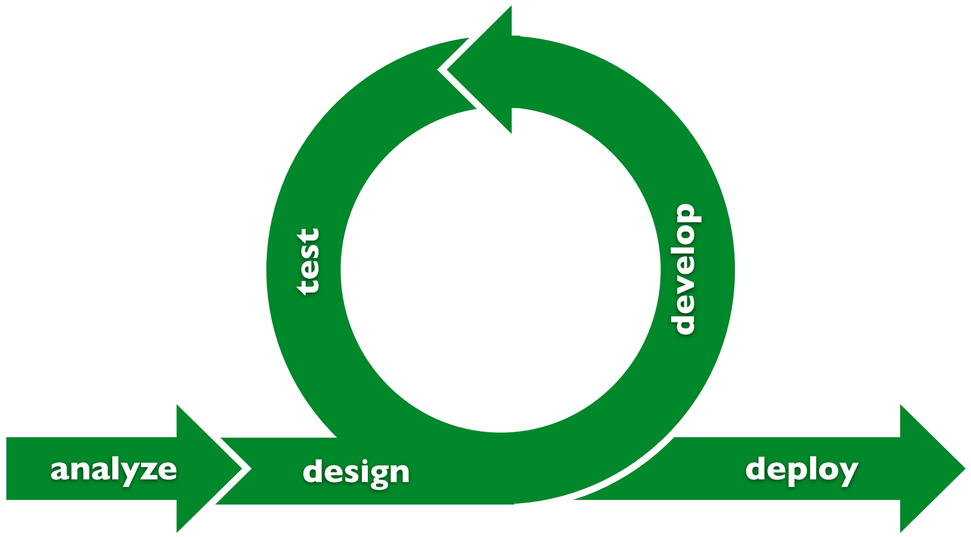


let's build together, step by step

the { iterative incremental } approach

several variants

- agile software development
- scrum process framework
- extreme programming



the agile manifesto

people & interactions
(vs. processes & tools)

working software
(vs. documentation)

customer collaboration
(vs. contract negotiation)

responding to change
(vs. following the plan)

the scrum framework

Inputs from Customers, Team, Managers, Execs

Product Owner

The Team

Scrum Master

Burn Down/Up Chart

Daily Standup Meeting

24 Hour Sprint

1-4 Week Sprint

Sprint Review

1
2
3
4
5
6
7
8

Prioritized list of what is required: features, bugs to fix...

Product Backlog

Team selects starting at top as much as it can commit to deliver by end of Sprint

Sprint Planning Meeting

Task Breakout

Sprint Backlog

Sprint end date and team deliverable do not change

Finished Work

Sprint Retrospective

source: https://www.neonrain.com/agile-scrum-web-development

development methodologies

waterfall vs. iterative

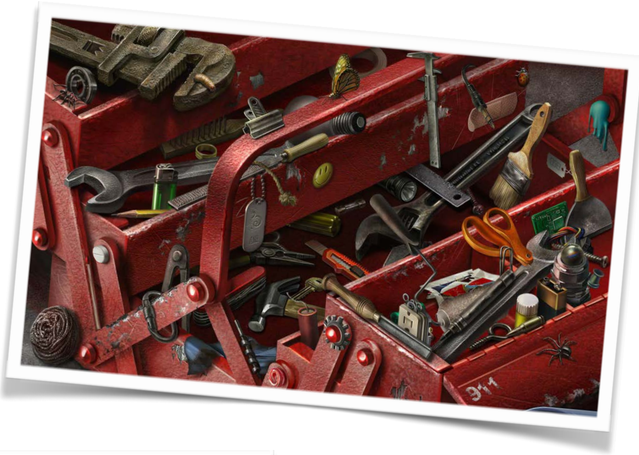
critical outlook

value-driven approaches iterative & agile	plan-driven approaches sequential & waterfall	formal method approaches math-based & determinism
marginally critical software	highly critical software	extremely critical software
requirements change often	requirements don't change often	strict and limited requirements
small team of developers	large team of developers	developers who can formally model requirements
culture responsive to changes	culture demanding structure	culture demanding extreme quality

extreme programming

FEEDBACK CULTURE	
embedded customer	customers available at all times to set priorities, define requirements, answer questions
user stories	planning based on brief user stories defined by customers to capture desired features
pair programming	code written by programmers working in pairs on a single computer to foster high quality
test automation	code thoroughly tested via automatic unit tests written before the actual code itself
CONTINUOUS PROCESS	
small releases	new versions of the software released frequently, incrementally delivering value to customers
continuous integration	complete software builds generated several times a day to avoid big integration problems later
regular refactoring	code incrementally improved by regular refactoring, without changing its external behavior
SHARED UNDERSTANDING	
simple design	adoption of the kiss principle at all times via refactoring if needed (kiss = keep it simple, stupid)
shared metaphor	shared understanding via a metaphor of the software, leading to consistent a naming scheme
collective ownership	responsibility of all the code shared by all the developers, meaning anyone can change anything
coding standards	consistent coding style and format throughout the code base, allowing for easy code sharing
PROGRAMMER WELFARE	
sustainable workload	awareness that coding is an intense activity, thus limiting work time to 40 hours/week

development practices & tools



coding

debugging

unit testing*

refactoring

versioning

user stories

planning

devops

profiling

continuous integration

*will be discussed next week

coding tools

HelloWorld.java — ~/SwitchDrive/Teaching/Courses/Current/Fall/Software Architecture/Code/work

HelloWorld.java

```
1  public class HelloWorld {
2    public static void main(String args[]) {
3          if (args.length > 0) {
4            for (String person : args) {
5              System.out.println("Hello " + person + "!");
6            }
7          } else {
8            System.out.println("Hello World!");
9          }
10   }
11 }
12
```

0 ▲ 0 HelloWorld.java 5:57 LF UTF-8 Java GitHub Git (0) 1 update

code — -bash — ttys001

```
wallace-palace:code garbi$ ls -l
total 8
-rw-r--r--@ 1 garbi  staff  287 Oct  3 09:39 HelloWorld.java
wallace-palace:code garbi$ javac HelloWorld.java
wallace-palace:code garbi$ ls -l
total 16
-rw-r--r--  1 garbi  staff  1002 Oct  3 11:01 HelloWorld.class
-rw-r--r--@ 1 garbi  staff   287 Oct  3 09:39 HelloWorld.java
wallace-palace:code garbi$ java HelloWorld
Hello World!
wallace-palace:code garbi$ java HelloWorld Frodo Sam Gandalf
Hello Frodo!
Hello Sam!
Hello Gandalf!
wallace-palace:code garbi$
```

editor, compiler, interpreter

command-line interface (shell)

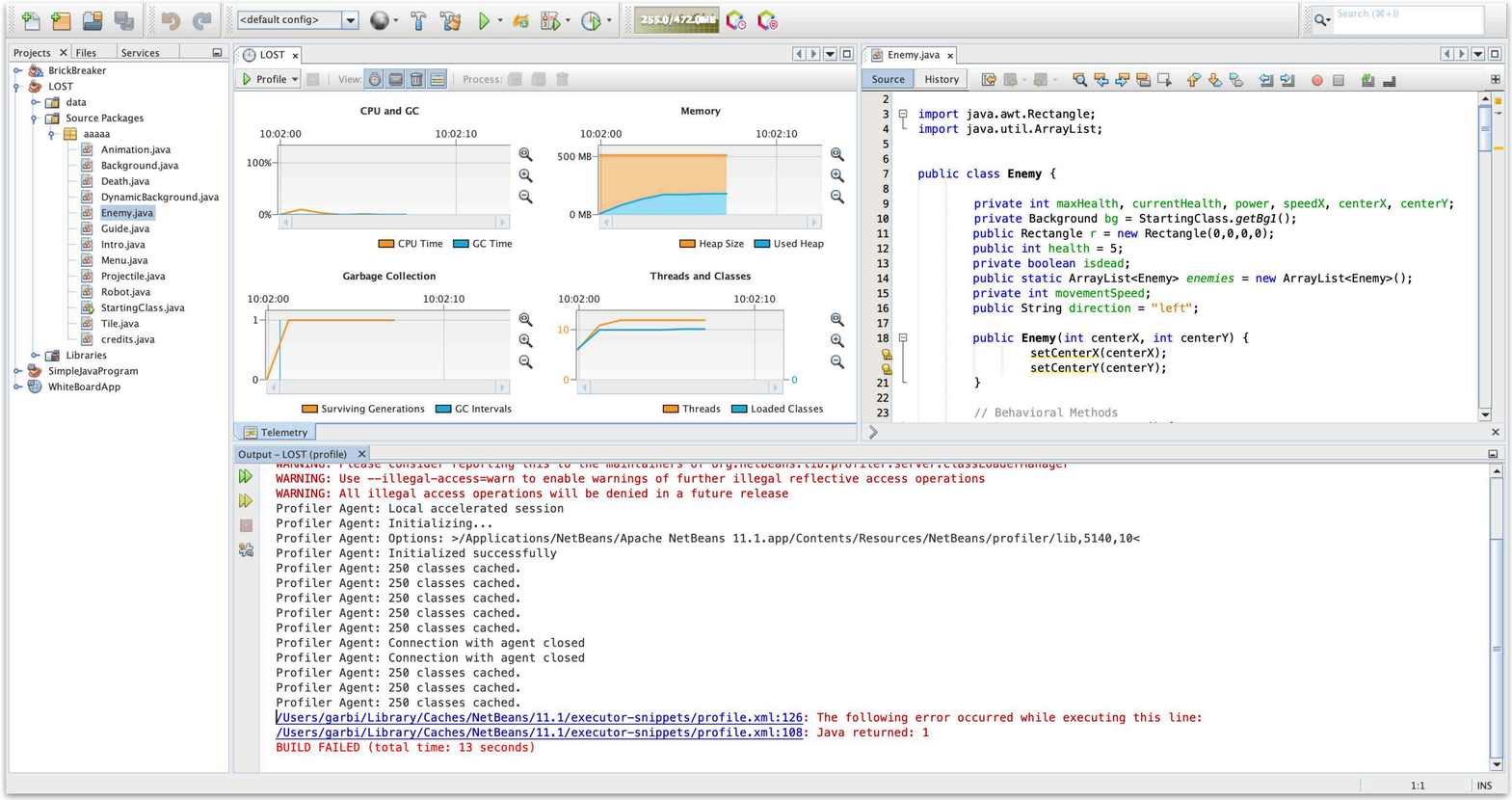
integrated
development
environments

Netbeans

PyCharm

Xcode

IntelliJ



debugging tools

HelloWorld.java — ~/SwitchDrive/Teaching/Courses/Current/Fall/Software Architecture/Code/CLI/logging

HelloWorld.java

```java
package ch.unil.doplab;

import java.util.logging.Logger;

/**
 *
 * @author garbi
 */
public class HelloWorld {

    private static final Logger logger = Logger.getLogger(HelloWorld.class.getName());

    public static void main(String args[]) throws InterruptedException {
        logger.info("I just entered the main() method");
        if (args.length == 0) {
            System.out.println("Hello World!");
            logger.severe("Ah Ah, nobody to greet?");
        } else {
            logger.info("Okay, I have people to greet!");
            for (String name : args) {
                System.out.println("Hello " + name + "!");
            }
        }
        System.out.println("Bye everbody!");
    }
}
```

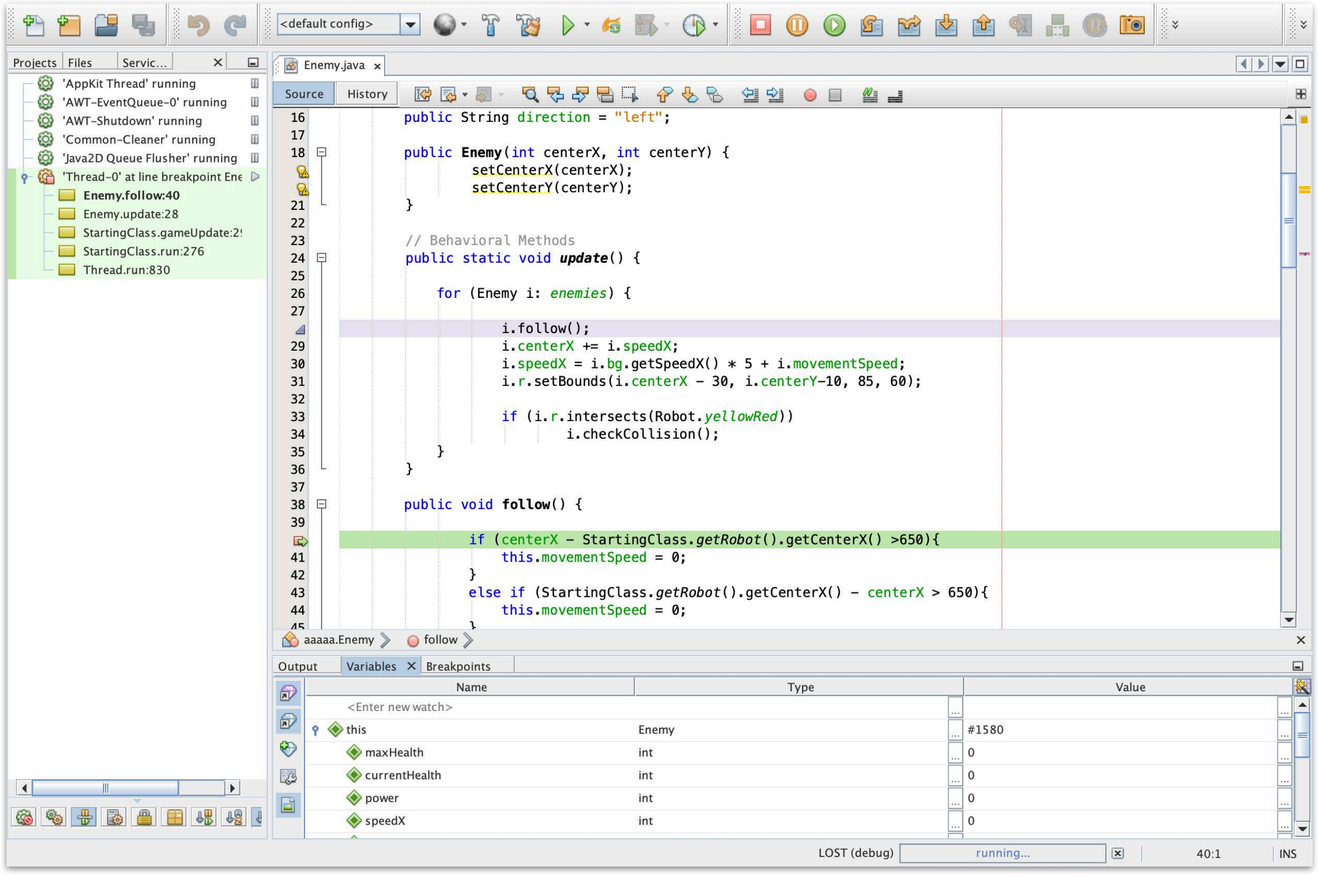
0 0 ~/SwitchDrive/Teaching/Courses/Current/Fall/Software Architec LF UTF-8 Java GitHub Git (0)

logging — -bash — ttys001

```
wallace-palace:logging garbi$ java ch.unil.doplab.HelloWorld
Oct 03, 2019 11:14:36 AM ch.unil.doplab.HelloWorld main
INFO: I just entered the main() method
Hello World!
Oct 03, 2019 11:14:36 AM ch.unil.doplab.HelloWorld main
SEVERE: Ah Ah, nobody to greet?
Bye everbody!
wallace-palace:logging garbi$
wallace-palace:logging garbi$ java ch.unil.doplab.HelloWorld Frodo Sam Gandalf
Oct 03, 2019 11:14:39 AM ch.unil.doplab.HelloWorld main
INFO: I just entered the main() method
Oct 03, 2019 11:14:39 AM ch.unil.doplab.HelloWorld main
INFO: Okay, I have people to greet!
Hello Frodo!
Hello Sam!
Hello Gandalf!
Bye everbody!
wallace-palace:logging garbi$
```

- trace via standard output (bad)
- trace via logging framework (good)

- symbolic debugger
  - ☑ breakpoints
  - ☑ step-by-step execution
  - ☑ examine variables in memory
  - ☑ on-the-fly bug correction



versioning tools

- at the **software level**, versions reflects its incremental nature

7.3.8

major version
big new feature
might break compatibility

minor version
small new features
no break in compatibility

patch version
only bug fixes
no break in compatibility

- at the **source code level**, versioning is tool to keep track of incremental changes and to make it possible to go back to a previously working version

Bazaar

mercurial

SUBVERSION

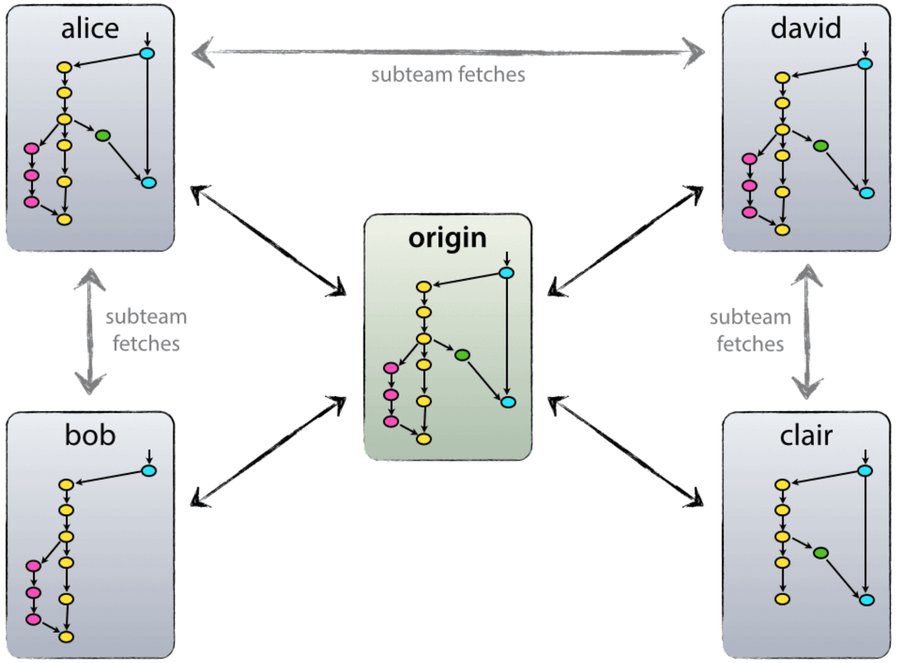
git

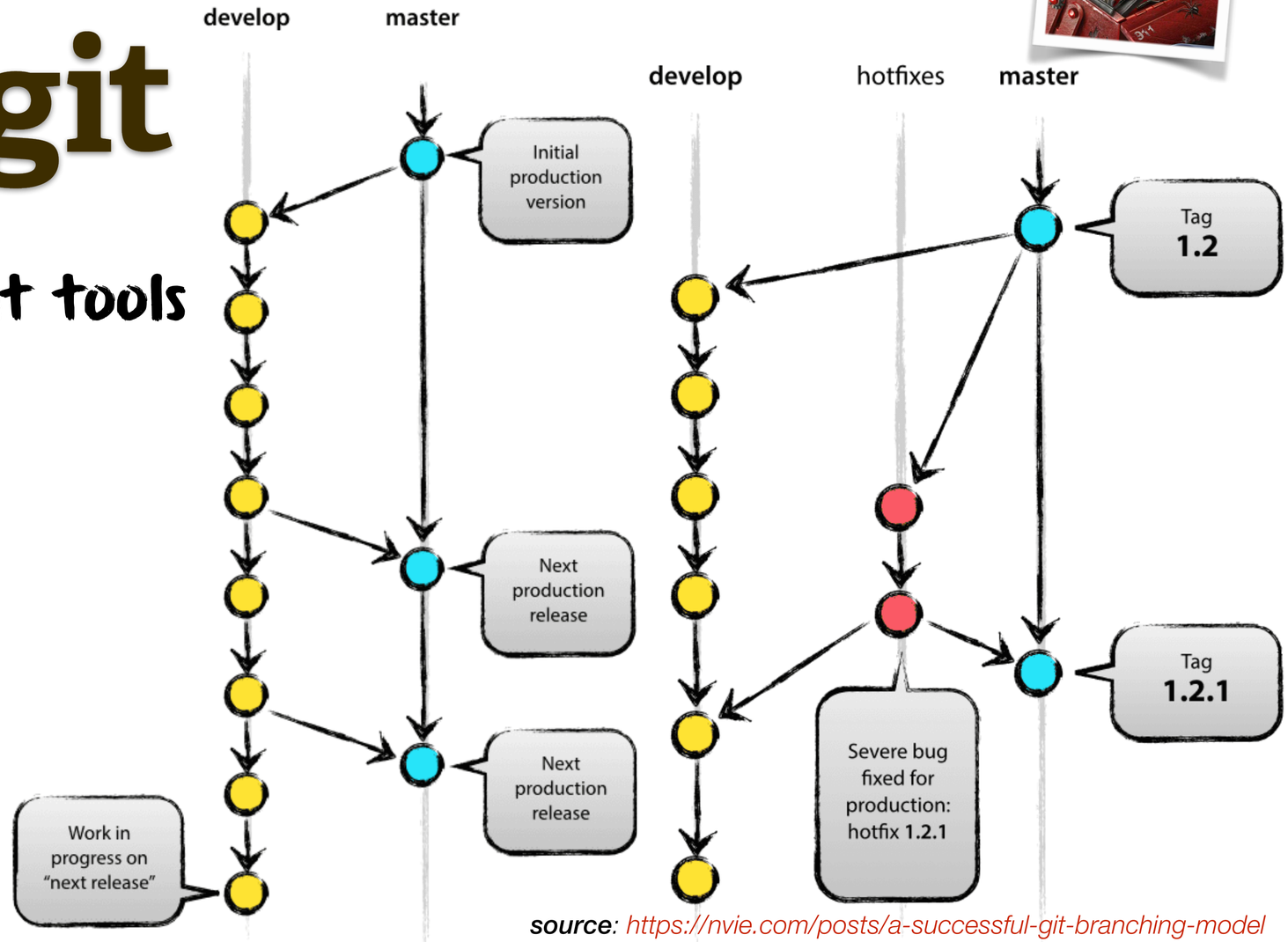
versioning tools

at the source code level, versioning is tool to keep track of incremental changes and to make it possible to go back to a previously working version

- ☑ command line tool
- ☑ local and distributed
- ☑ rich branching model
- ☑ available in visual development tools

git





source: https://nvie.com/posts/a-successful-git-branching-model

refactoring tools

- refactoring is the process of restructuring existing code without changing its external behavior
- refactoring aims at reducing the complexity, improving readability, in order to increase software maintainability and extensibility
- refactoring tools are usually embedded in development software
- refactoring typically include changing class or method names, extracting interfaces and superclasses,

Refactoring

Rename **Projectile** to **Arrow** [12 occurrences]
- LOST
  - StartingClass.java
    - Update reference to Projectile
    - Update reference to Projectile
    - Update reference to Projectile
    - Update reference to Projectile
  - Robot.java
    - Update reference to Projectile
    - Update reference to Projectile
    - Update reference to Projectile
    - Update reference to Projectile
    - Update reference to Projectile
  - Projectile.java
    - Change class name
    - Rename method
    - Rename file Projectile.java

Robot.java 2/4 Refactored Robot.java

```
        if (isMovingRight() == false && isMovingLeft() == true) {
            moveLeft();
        }

        if (isMovingRight() == true && isMovingLeft() == false) {
            moveRight();
        }

    }

    public void jump() {
        if (jumped == false) {
            speedY = JUMPSPEED;
            jumped = true;
        }

    }

    public void shoot() {
        Projectile p;
        if (getDirection() == "right")
            p = new Projectile(centerX-5, centerY-5,true);
        else
            p = new Projectile(centerX-5, centerY-5,false);
        projectiles.add(p);
    }

    public int getCenterX() {
        return centerX;
```

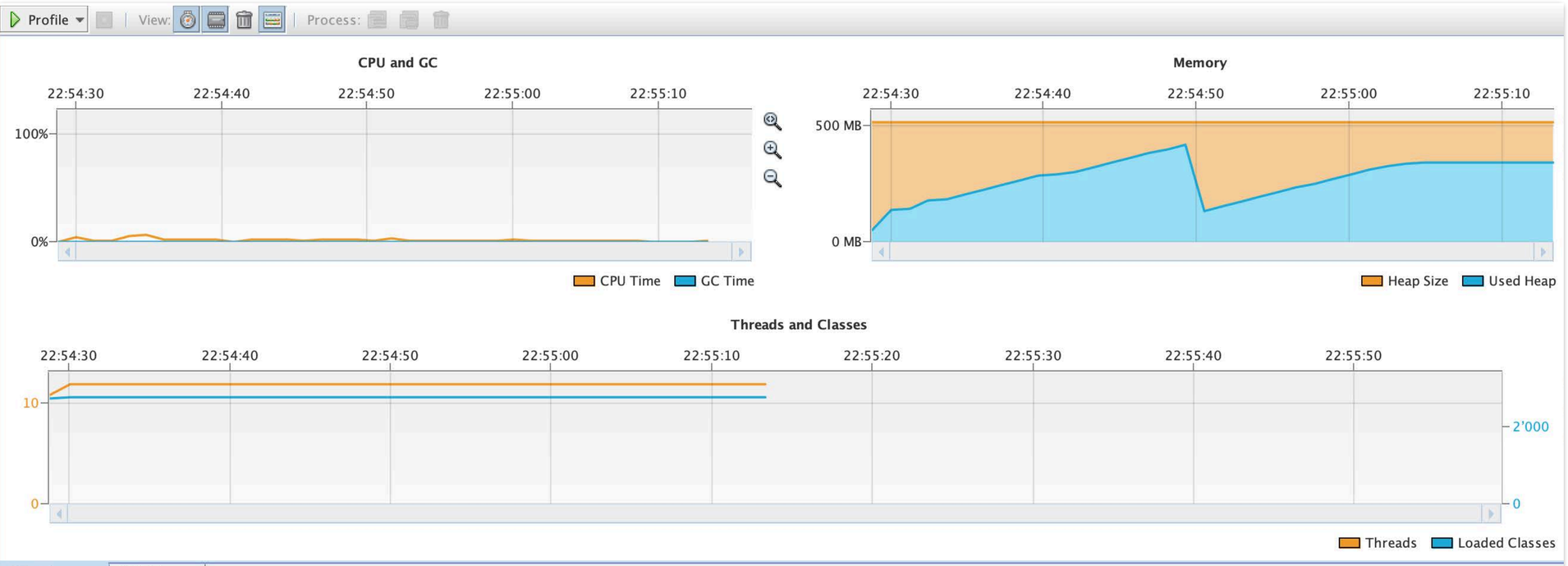
```
        if (isMovingRight() == false && isMovingLeft() == true)
            moveLeft();
        }

        if (isMovingRight() == true && isMovingLeft() == false)
            moveRight();
        }

    }

    public void jump() {
        if (jumped == false) {
            speedY = JUMPSPEED;
            jumped = true;
        }

    }

    public void shoot() {
        Arrow p;
        if (getDirection() == "right")
            p = new Arrow(centerX-5, centerY-5,true);
        else
            p = new Arrow(centerX-5, centerY-5,false);
        projectiles.add(p);
    }

    public int getCenterX() {
        return centerX;
```

The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil (or at least most of it) in programming.

Donald Knuth, The Humble Programmer. Communication of the ACM, vol. 17, no. 12. December 1974. Turing Award Lecture.

profiling tools

- diagnosing performance issues is counter-intuitive
- profiling consists in dynamically analysing the resource usage of a program
- profiling tool instrument the source or binary code of the program



Name	Live Bytes		Live Objects	
java.awt.geom.**AffineTransform**	109'586'808 B	(31.4%)	1'522'039	(44.2%)
double[]	97'386'160 B	(27.9%)	1'522'044	(44.2%)
byte[]	72'719'296 B	(20.8%)	44'940	(1.3%)
int[]	56'355'952 B	(16.1%)	1'852	(0.1%)
java.lang.ref.**WeakReference**	5'163'168 B	(1.5%)	161'349	(4.7%)
java.lang.**String**	1'055'544 B	(0.3%)	43'981	(1.3%)
java.util.**HashMap$Node**	886'976 B	(0.3%)	27'718	(0.8%)
java.awt.**Rectangle**	874'016 B	(0.3%)	27'313	(0.8%)
aaaaa.**Tile**	832'416 B	(0.2%)	17'342	(0.5%)
java.lang.**Class**	402'432 B	(0.1%)	3'284	(0.1%)
java.security.**AccessControlContext**	371'080 B	(0.1%)	9'277	(0.3%)
long[]	334'848 B	(0.1%)	124	(0%)
sun.java2d.**SunGraphics2D**	321'984 B	(0.1%)	1'548	(0%)
char[]	297'744 B	(0.1%)	35	(0%)
java.lang.**Object[]**	295'040 B	(0.1%)	4'712	(0.1%)
java.util.**HashMap$Node[]**	284'744 B	(0.1%)	528	(0%)
sun.java2d.pipe.**Region**	205'520 B	(0.1%)	5'138	(0.1%)
java.lang.reflect.**Method**	87'912 B	(0%)	999	(0%)
java.awt.**Point**	79'104 B	(0%)	3'296	(0.1%)
java.awt.geom.**Rectangle2D$Double**	74'304 B	(0%)	1'548	(0%)
java.util.**ArrayList$Itr**	73'920 B	(0%)	2'310	(0.1%)
java.util.concurrent.**ConcurrentHashM**	70'304 B	(0%)	2'197	(0.1%)
java.awt.event.**MouseEvent**	54'560 B	(0%)	682	(0%)
java.awt.event.**InvocationEvent**	49'600 B	(0%)	775	(0%)
sun.awt.**EventQueueItem**	48'504 B	(0%)	2'021	(0.1%)
sun.lwawt.macosx.**NSEvent**	45'360 B	(0%)	567	(0%)
java.lang.**Class[]**	41'992 B	(0%)	1'685	(0%)
java.util.**ArrayList**	40'536 B	(0%)	1'689	(0%)