Software Engineering
The University of Auckland

Quality Assurance
Introduction to Part II

Part II - Lecture 1

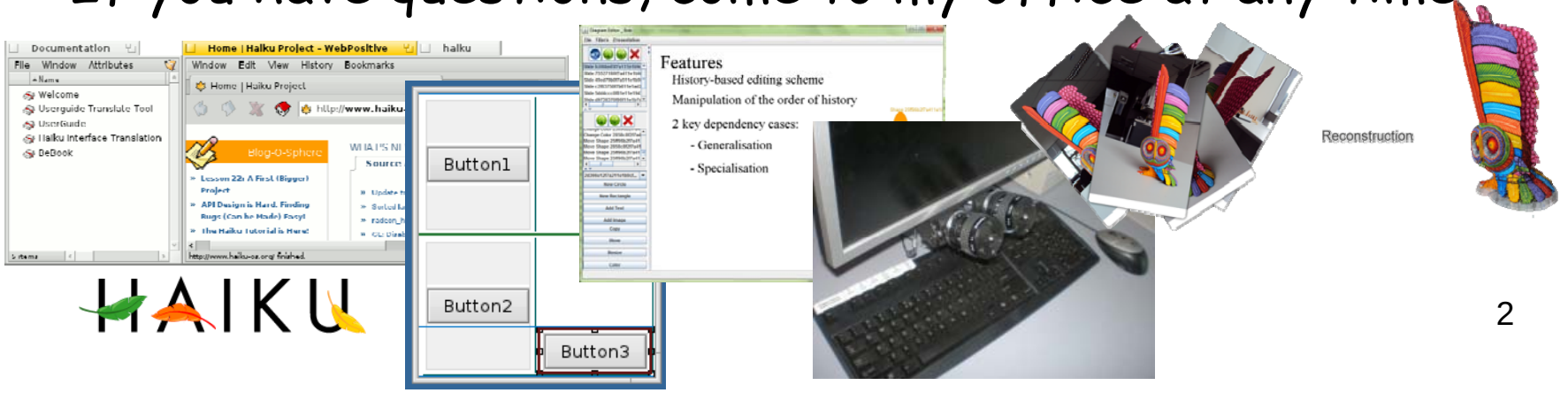
Christof Lutteroth

Software Engineering
The University of Auckland

- From Berlin, Germany
- Some years ago this was the first lecture given by me (you can watch the old lectures on video)
- My research interests: HCI, SE tools, …
- Contact details:

  christof@cs.auckland.ac.nz
  Phone 373-7599 84478
  Office: 303 - 494 (4th level CompSci building)

- If you have questions, come to my office at any time



Software Engineering
The University of Auckland

Introduction to Part II

Quality means doing it right when no one is looking
(Henry Ford)

Why not just Dynamic Detection of Defects?

- First half mostly about dynamic detection of defects
- Can be time consuming, costly, complicated…but is necessary!!!
- Requires a (partially) running system already. What if we don't have it yet?
- Usually in the implementation stage of a project. What if the defect has its origin in the specification or design?
- What if our test cases contain defects?

How can we avoid defects in the first place?

Static Detection of Defects

- If we cannot avoid defects, detect them as early as possible
- "Static" means before runtime
- Most compilers detect many type errors (type systems), other tools detect even more errors
- The more specification we give, the more we can verify; formal specification languages (e.g. Z, Alloy, Hoare Calculus)
- Sad result from theory:
  some errors cannot be detected statically!!!
- But we can be "conservative":
  prohibit even code that **might** be an error

Best Practices

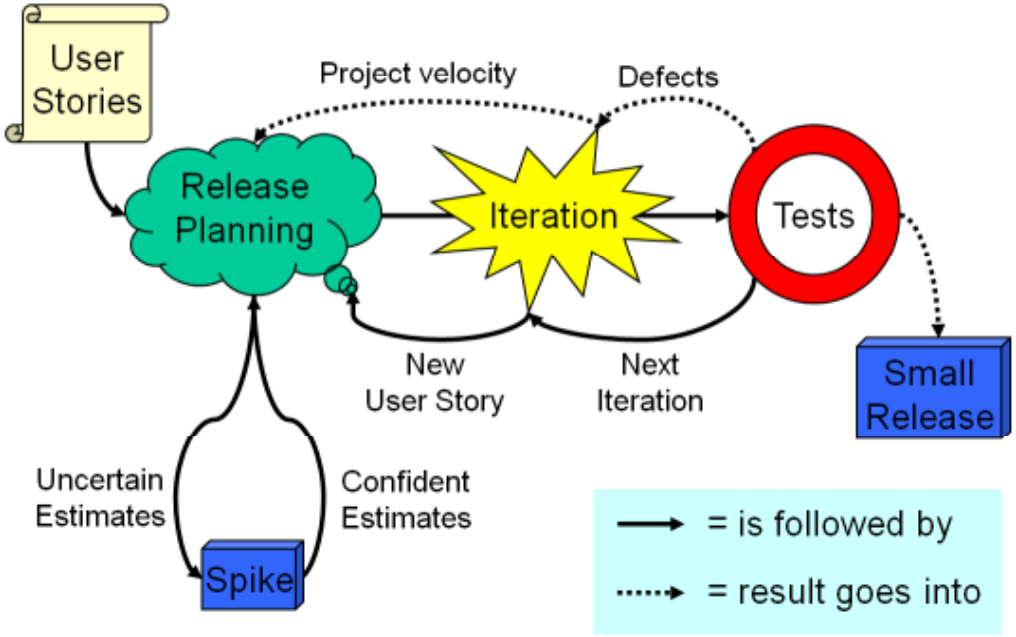
- Avoiding the mistakes that cause defects by "doing the right thing"
  - **Software development processes**:
    how to undertake a software project
  - **Software tools**:
    use the best suitable tools to get the best results
  - **Design & coding guidelines**:
    how to avoid design/programming mistakes
- Learn from other's experience & mistakes
  (not as effective as learning from own mistakes, but saves time ☺)
- But: what works for others does not necessarily work for you (nothing can replace your own experience)

Software Engineering
The University of Auckland

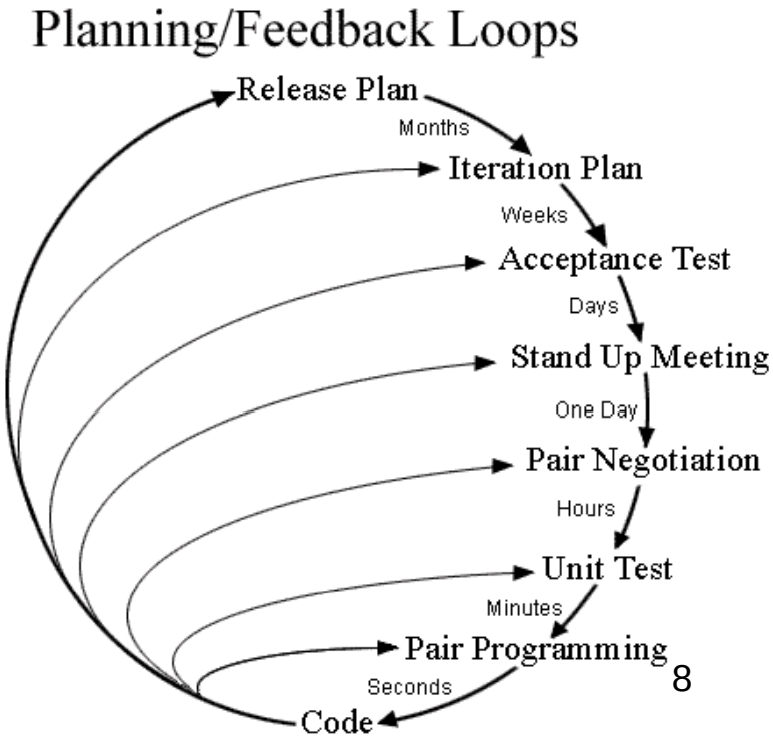
Software Development Processes

"I love it when a plan comes together! "

eXtreme Programming (XP)

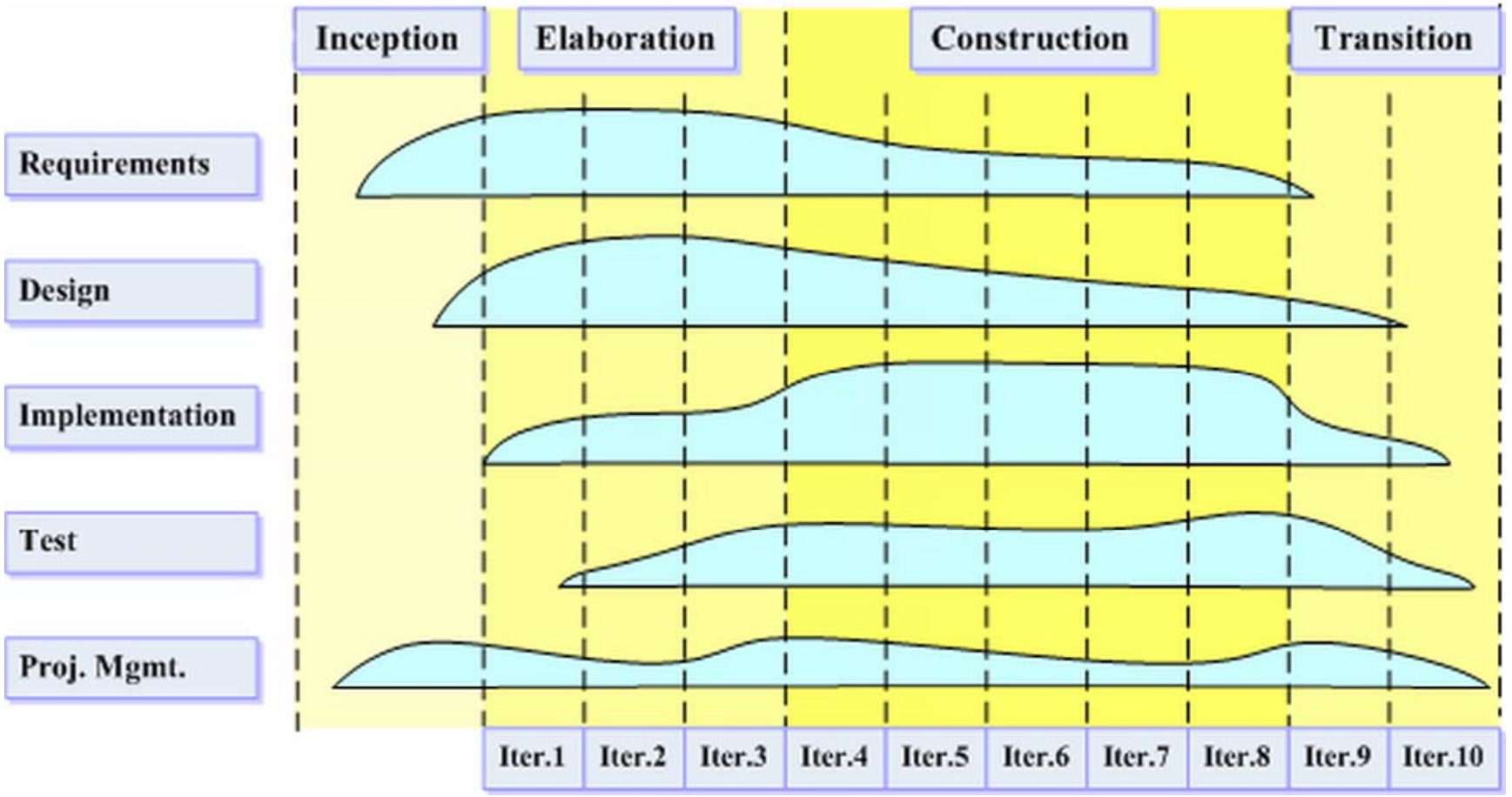


Don Wells



- Generic plan of how to conduct a project
- 12 best practices
- Relies heavily on feedback about the software

Rational Unified Process (RUP)



- Comprehensive process framework
- Used esp. For large projects

Software Engineering
The University of Auckland

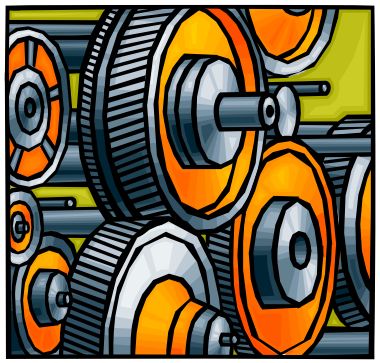
Software Development Tools

First we shape our tools and then our tools shape us
(Marshall McLuhan)

Software Tools

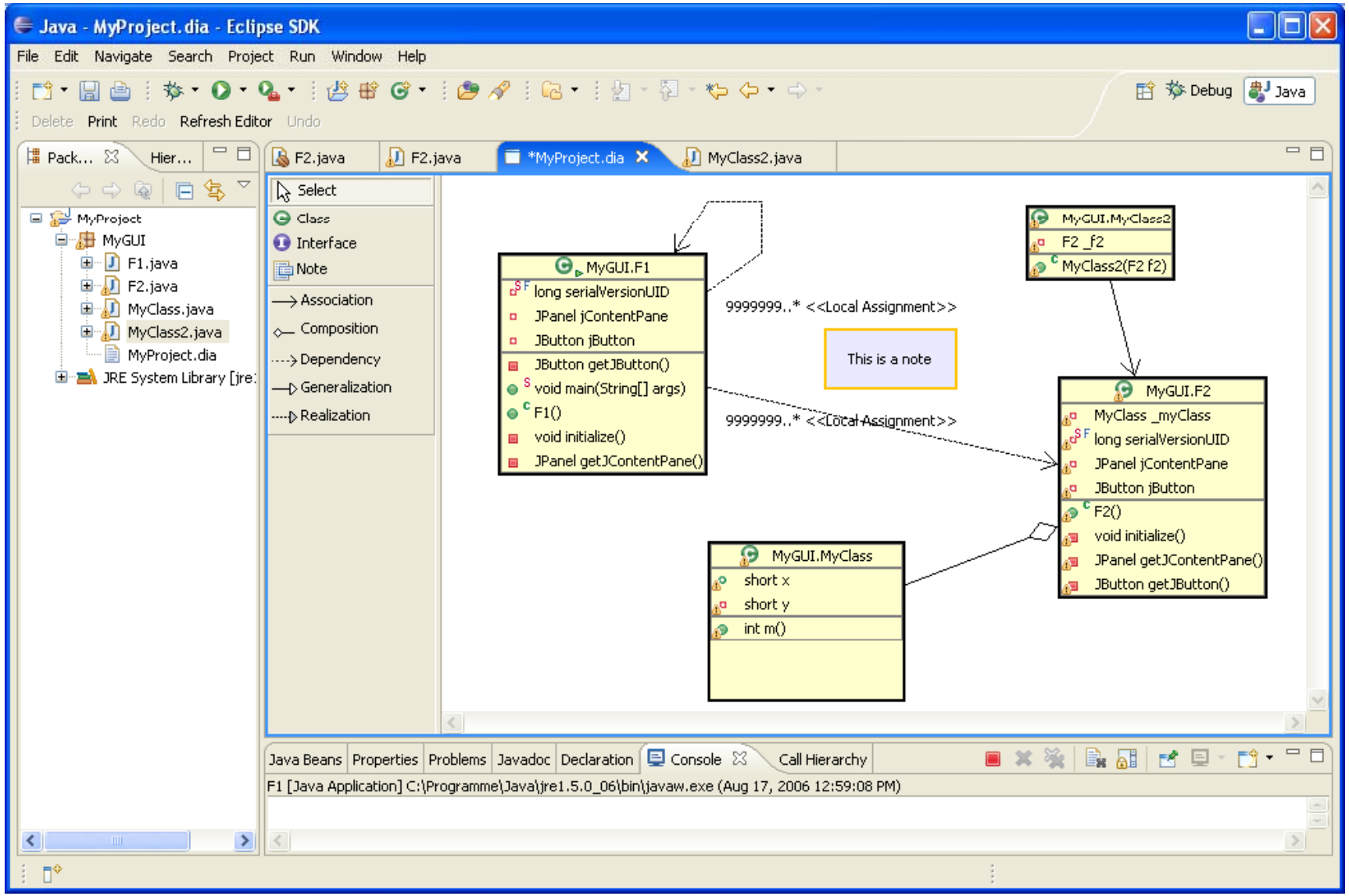
- Humans are necessary for creative, intelligent tasks
- Tools can **support** such tasks
  - Increase productivity with useful functionality
  - Guide the developer (e.g. context help)
  - Avoid defects

- Humans are not necessary for highly repetitive, routine work
- Tools can **automate** such tasks
  - Increase productivity; more time for creative work
  - Avoid defects introduced by the human factor



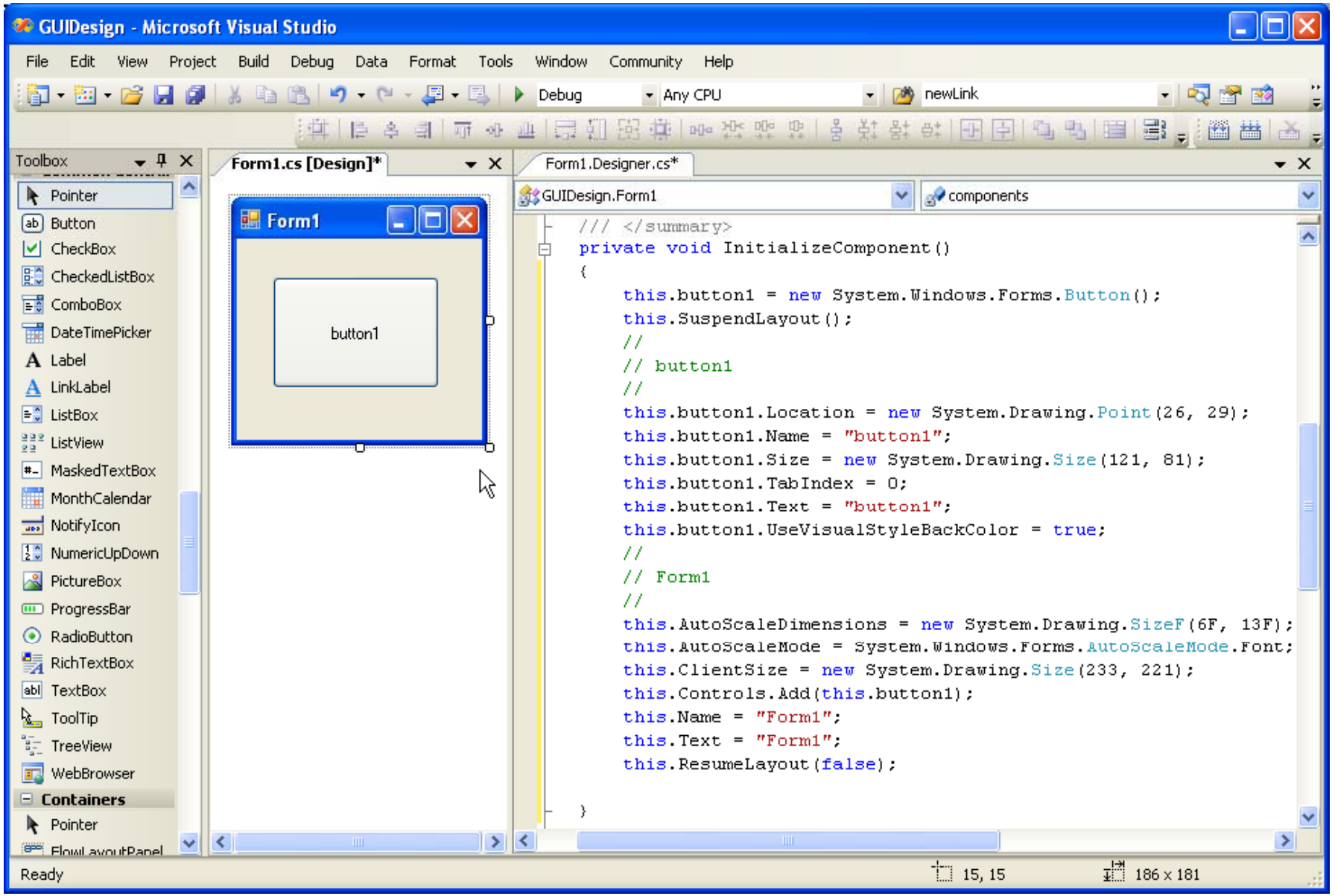
Modeling tools

Software Engineering
The University of Auckland



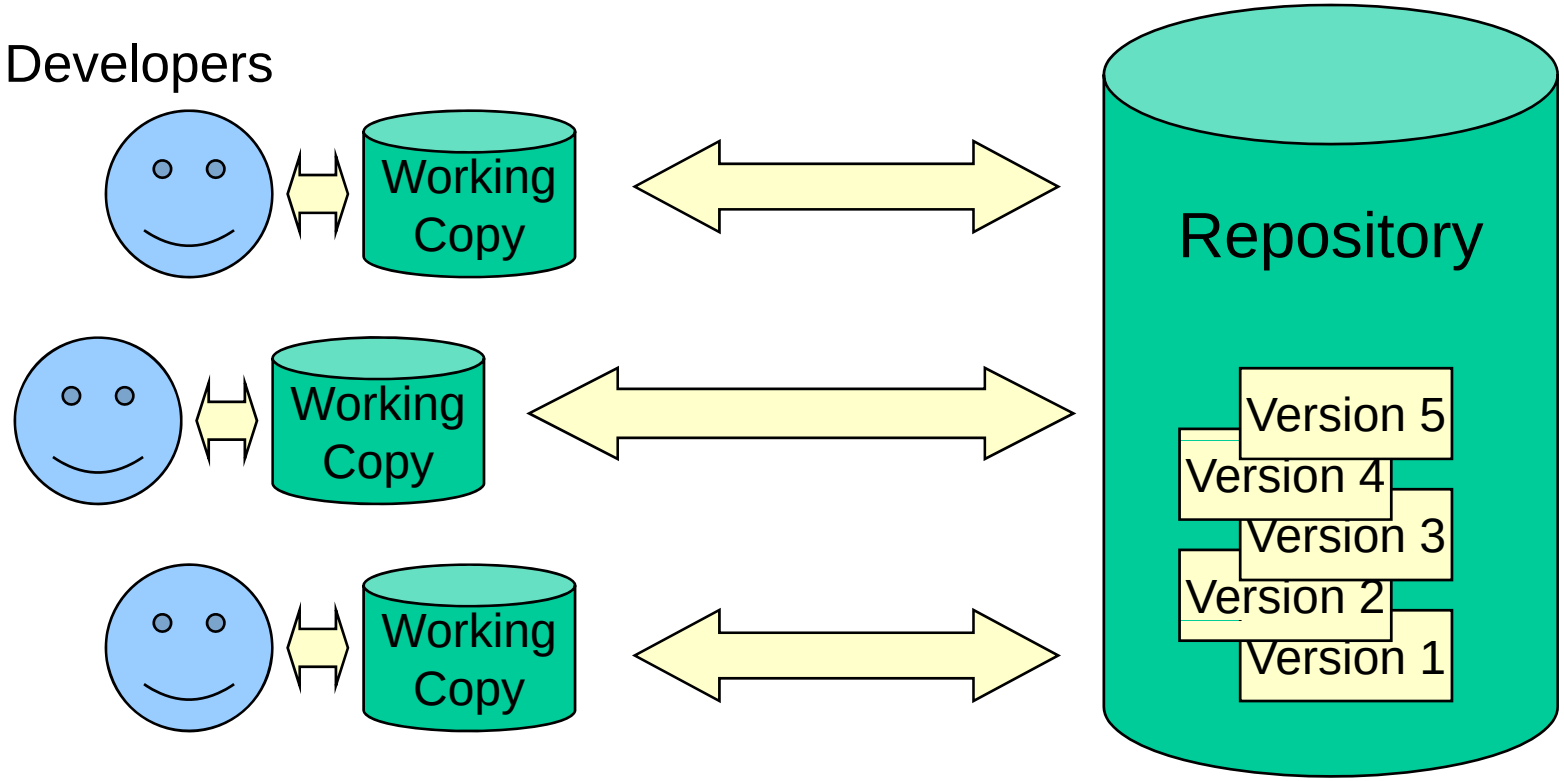
GUI Builders

Software Engineering
The University of Auckland

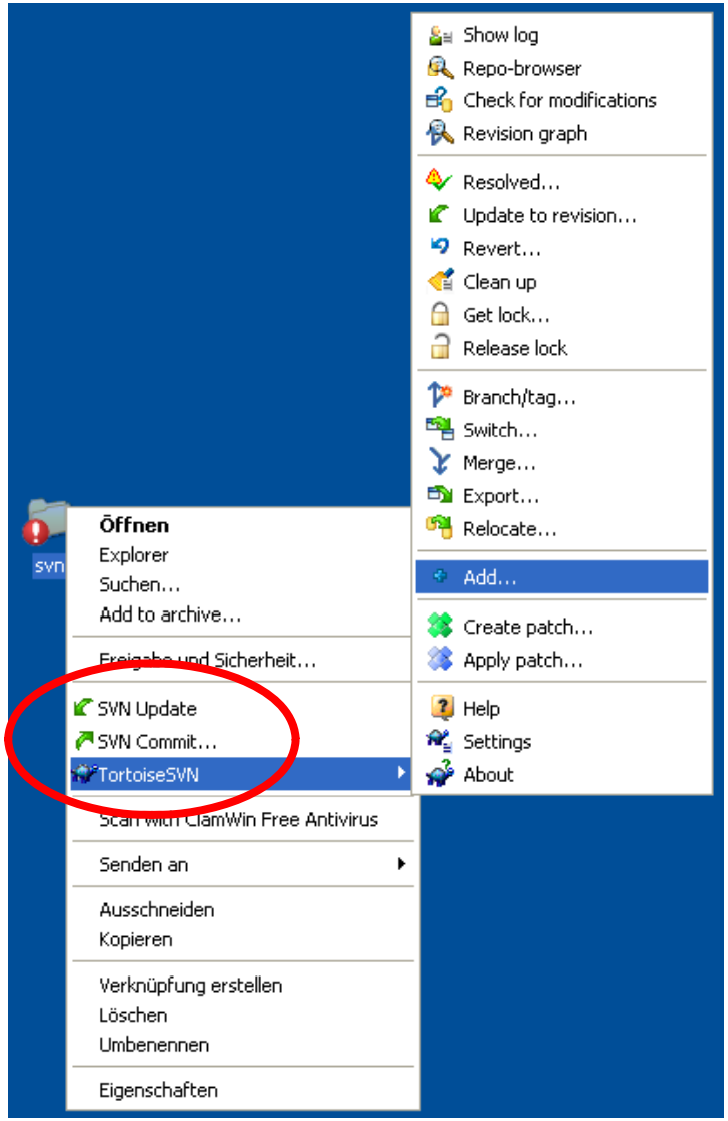


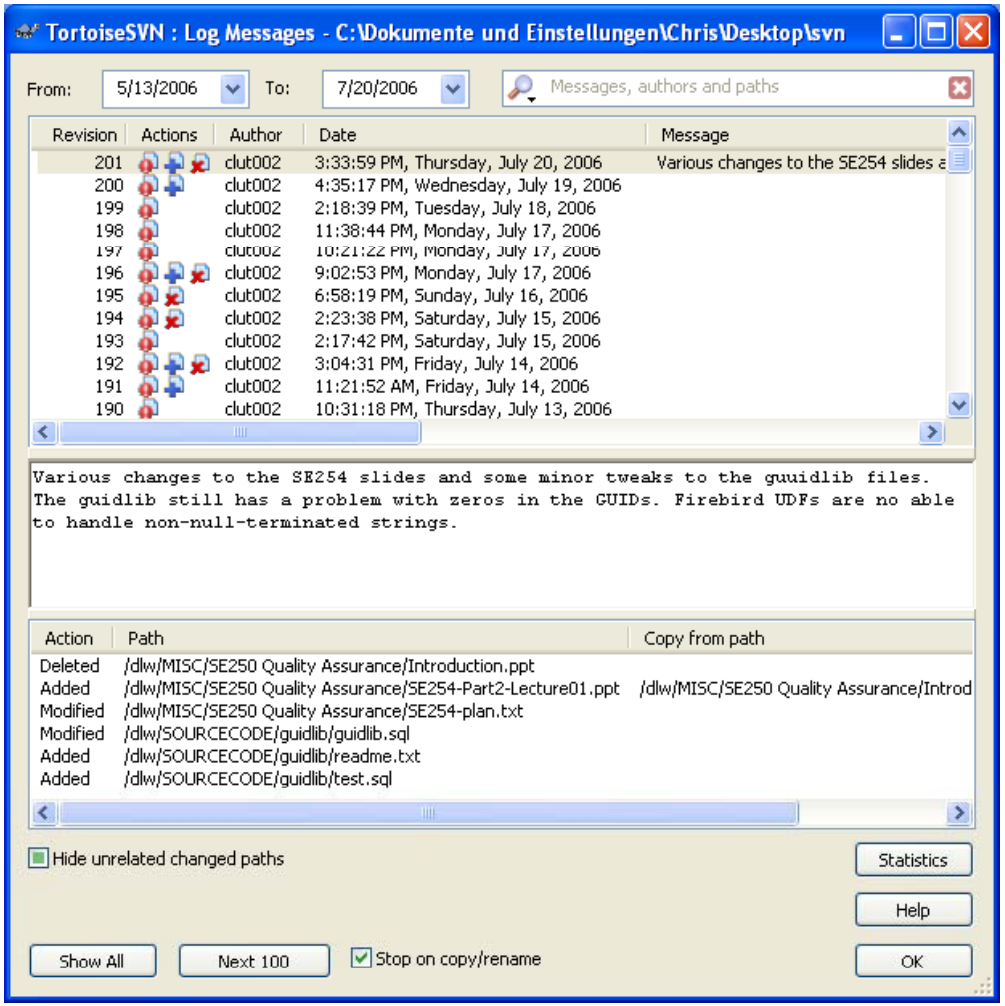
Version Control Systems

- Technology to manage changes that several developers do on a common repository
- Changes create new version of the changed files
- Old versions are always accessible



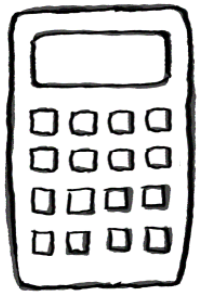
Version Control Systems
Example: Subversion



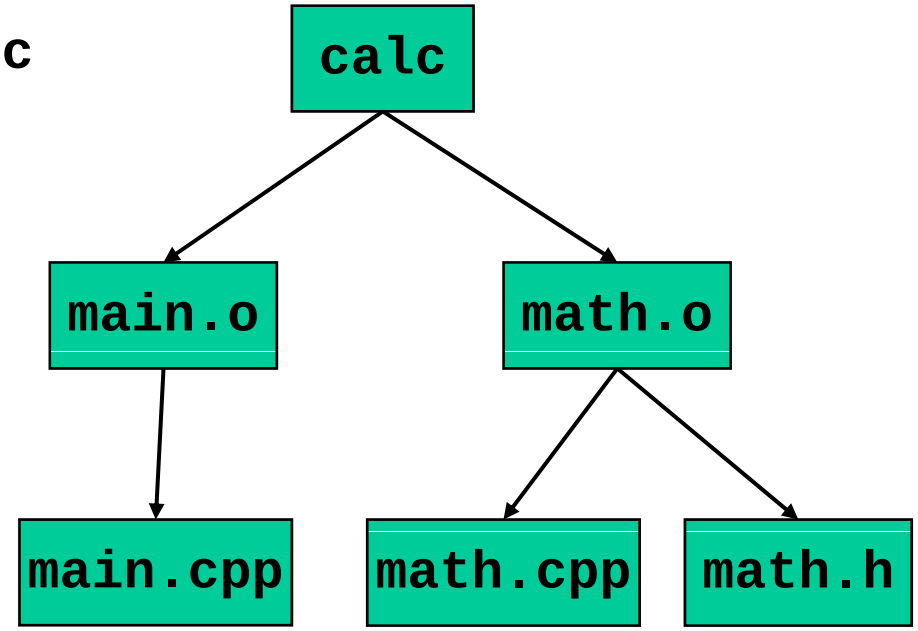


Build Tools
Example: make


```
all: calc

calc: main.o math.o
   g++ main.o math.o -o calc

main.o: main.cpp
   g++ -c main.cpp

math.o: math.cpp math.h
   g++ -c math.cpp

clean:
   rm *.o
```

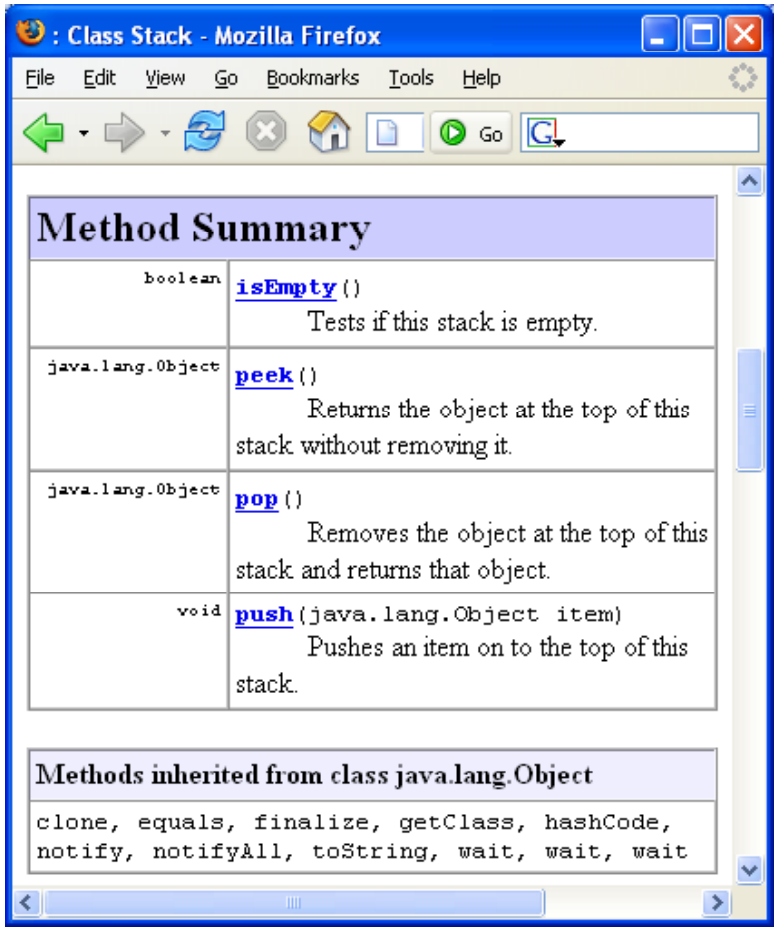


Dependency graph



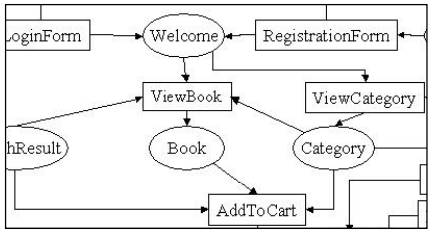
Documentation Tools Example: JavaDoc

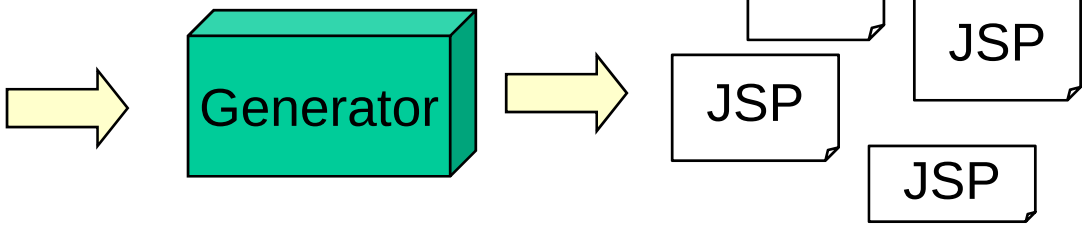

```
public class Stack {
  /**
   * Pushes an item on to
   * the top of this stack.
   * @param item the item to be pushed
   */
  public void push(Objec item){
        this.elements.add(item);}

  /**
   * Removes the object at the top
   * of this stack and returns that
   * object.
   * @return The object at the top
   *         of this stack.
   * @exception NoSuchElementException
   *         if this stack is empty.
   */
  public Object pop()
  throws NoSuchElementException {
    // ...
} }
```



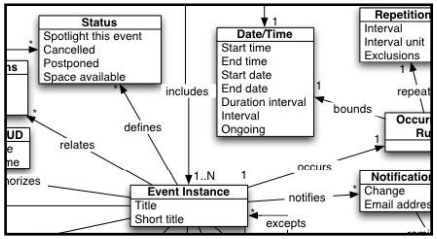
Generators

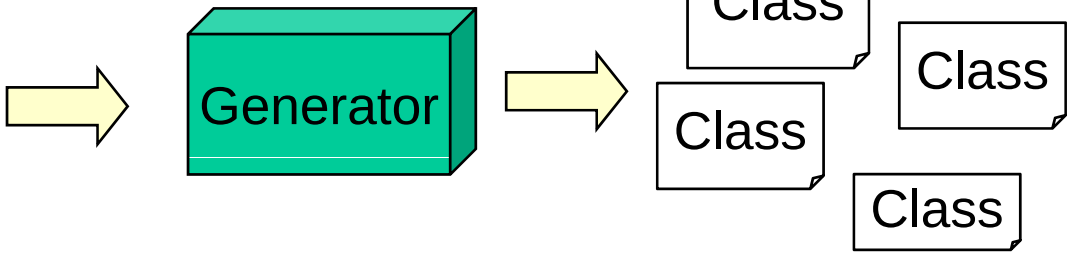
Generation of a web UI



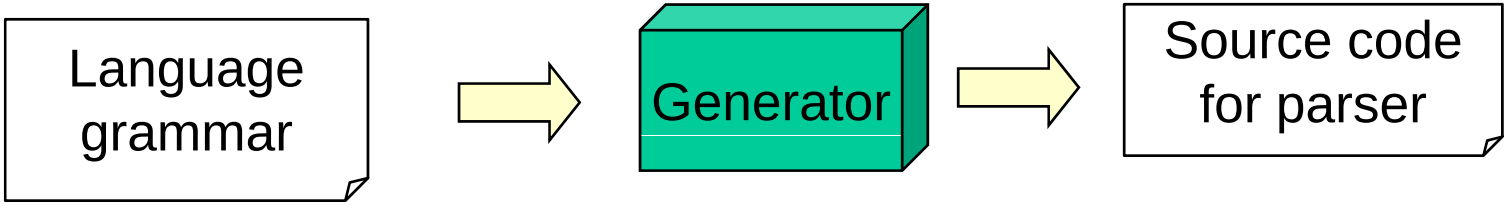


Generation of Java classes





Generation of language parsers

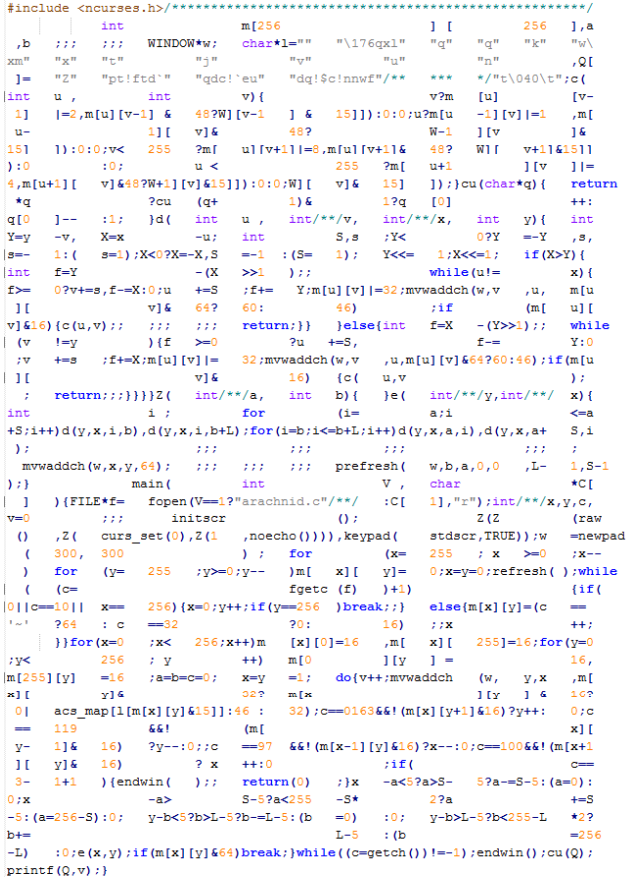


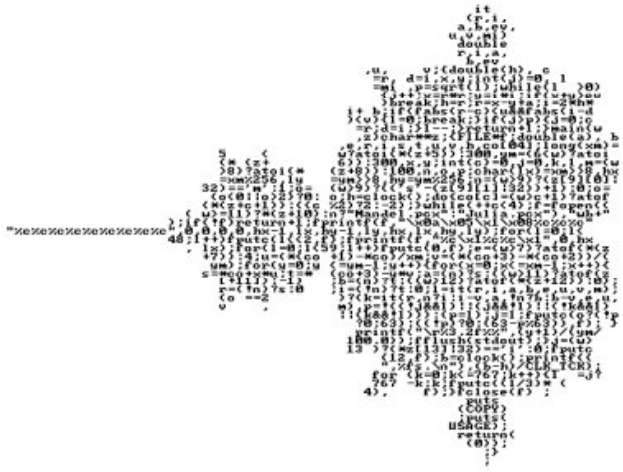
Software Engineering
The University of Auckland

Coding Principles

"Use the source, Luke"

Coding Style





```
main(a,b){
while((a=getchar())+1)
putchar((b=64^a&223)
&&b<27?a&96
|(b+12)%26+1:a);}
```

- International Obfuscated C Code Contest (iocccc.org)
- Coding style guidelines help to achieve clear, consistent code (esp. when many people are involved)

Coding Principles

Handling Exceptions

Interface
vs
Implementation

Information Hiding
(painting by Bev Doolittle)

Today's summary

- **Dynamic detection** of defects (e.g. testing, debugging) is necessary, but not enough
- **Static checking** (e.g. with type systems) can detect some defects before runtime
- We can avoid defects by following **best practices** of software development and using tools
- **Software tools** support the developer in performing creative tasks and can automate routine tasks

This week no tutorial, no lab.
Friday: Part 4 Exhibition!

Quiz

1. How can defects be avoided?
   Name and explain 3 examples.

2. What tools exist for supporting software development?
   Name 5 examples and explain what they do.