COLUMBIA UNIVERSITY
PROGRAMMING LANGUAGES AND TRANSLATORS
COMS W4115

SouL
SOUND LANGUAGE

DEVELOPERS

Andrew Goldin
Project Manager
adg2160@columbia.edu

Cindy Long
System Architect
xl2259@columbia.edu

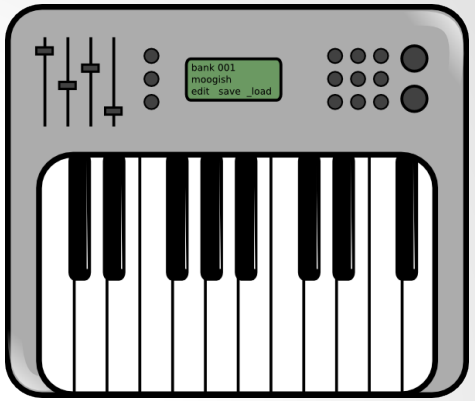
Matt Kim
System Integrator
mjk2189@columbia.edu

Kevin Walters
Language Guru
kmw2168@columbia.edu

SUPERVISOR
Professor Alfred Aho

DATE
May 16, 2013

WHAT IS SouL?



- **Sou**nd **L**anguage
- **SouL** is for **musicians**
  - From amateur to expert
  - From hobbyist to professional
- **SouL** was born out of a lack of an easier standard way to do MIDI-based music programming
- Some languages such as Java support MIDI functionality, but are not very intuitive

SouL
SOUND LANGUAGE

SouL IS SIMPLE

- Playing a note in Java:

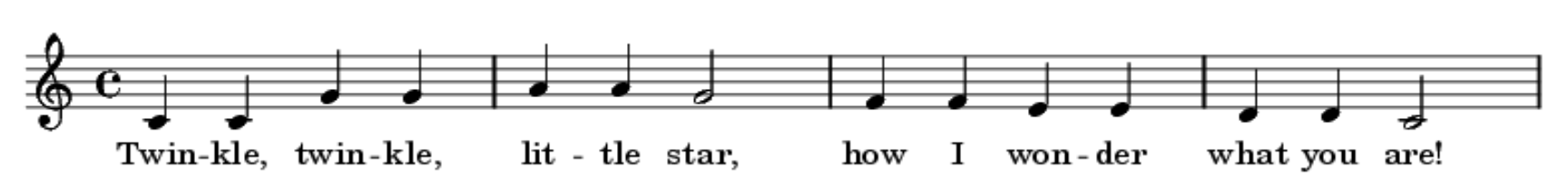
```
import javax.sound.midi.*;
public class PlayD {
    public static void main(String[] args) throws Exception {
        Sequence sequence = new Sequence(Sequence.PPQ, 16);
        Track track = sequence.createTrack();
        track.add(new MidiEvent(new ShortMessage(ShortMessage.NOTE_ON, 0, 62, 127), 0));
        track.add(new MidiEvent(new ShortMessage(ShortMessage.NOTE_OFF, 0, 62, 0), 64));
        Sequencer sequencer = MidiSystem.getSequencer();
        sequencer.open();
        sequencer.setSequence(sequence);
        sequencer.addMetaEventListener(new MetaEventListener() {
            public void meta(MetaMessage m) {
                if (m.getType() == 47) { System.exit(0); }
            }
        });
        sequencer.start();
    }
}
```

- Playing a note in SouL:

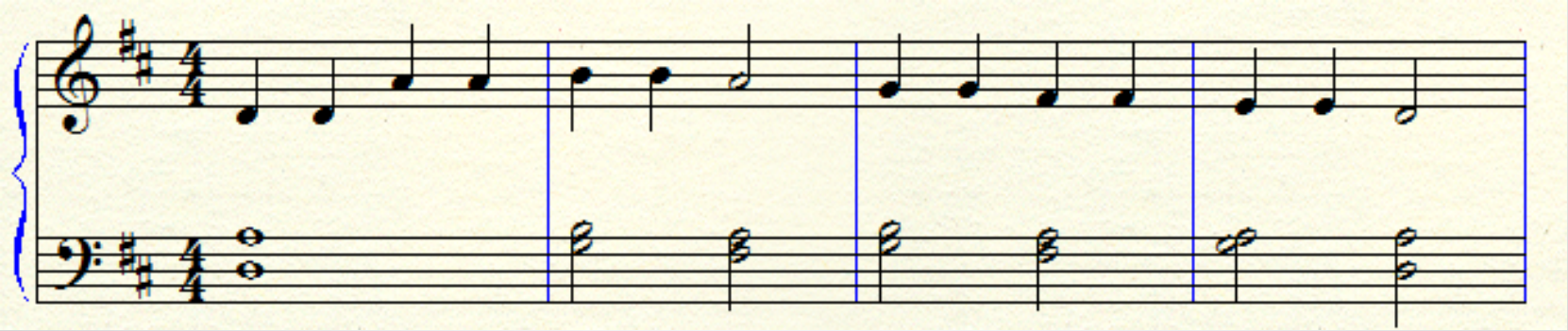
```
play(Note('D4', 127, WHOLE));
```

DEMOS

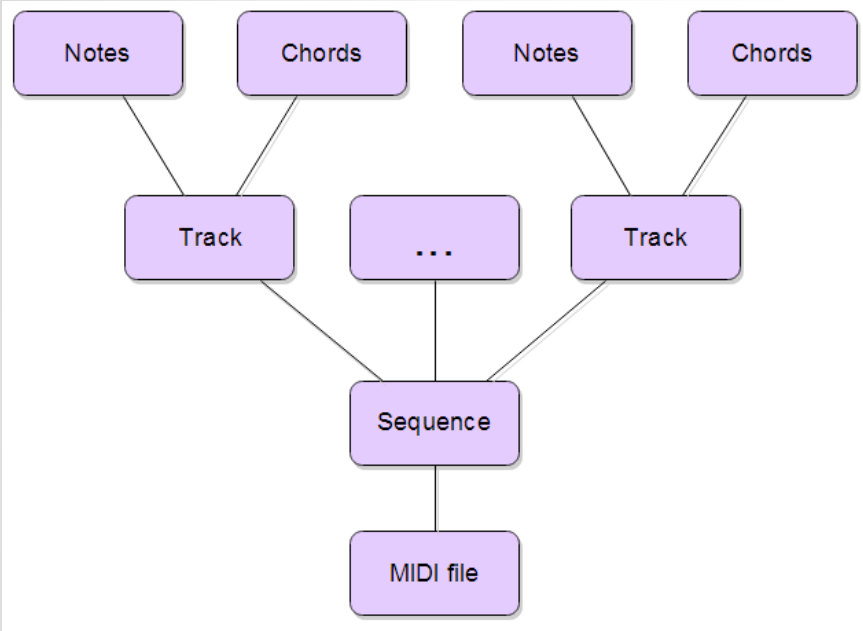
- Twinkle Twinkle Little Star



- Twinkle Twinkle Little Star 2.0



SYNTAX / MIDI STRUCTURE




```
Note n = Note('C4', 80, WHOLE);
Chord c = Chord(('C4', 'G4'), 80, WHOLE);

Track t = Track();
t.add(n); t.add(c);

Sequence s = Sequence();
s.add(t);

Midi m = Midi("MyFile.mid");
m.write(s);
```

- A **SouL program** revolves around manipulating these elements.

SYNTAX

- There are 33 **keywords** (i.e. `pitch, duration, decimal, play, print, WHOLE`)
  - They can be **types** or the names of **functions**
  - They can represent note **durations**
  - `true, false`, etc.

```
pitch p = 'C#5';
decimal d = 4.5;
instrument i = 40;
Midi m = Midi("test.mid");
duration wh = WHOLE;

while (p <= 'C#6') {
    play(Note(p, 127, wh));
    p += 2;
}
```

SouL
SOUND LANGUAGE

SYNTAX

- Java-like syntax, simplified:
  - Semicolons to end statements
  - Construction and manipulation of objects
  - Support of control-flow and arithmetic
  - Objects within objects
- There are currently 9 **built-in functions** that allow manipulation of objects
  - This is where SouL shines
- A look back at a SouL program - **Twinkle Twinkle Little Star**

SouL
SOUND LANGUAGE

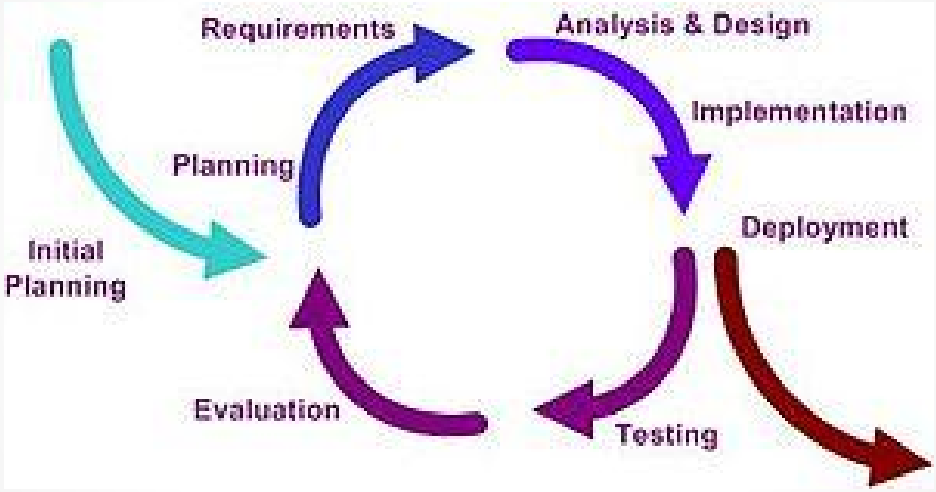
PROJECT MANAGEMENT

- Met early to decide language and roles
- Facebook and WhatsApp for online discussion, planning meetings
- Met once a week, more when deadlines approached
- Used Google Drive to keep all documents in one place, for real-time group editing
- Kept a meetings log to record weekly progress

SouL
SOUND LANGUAGE

PROJECT MANAGEMENT

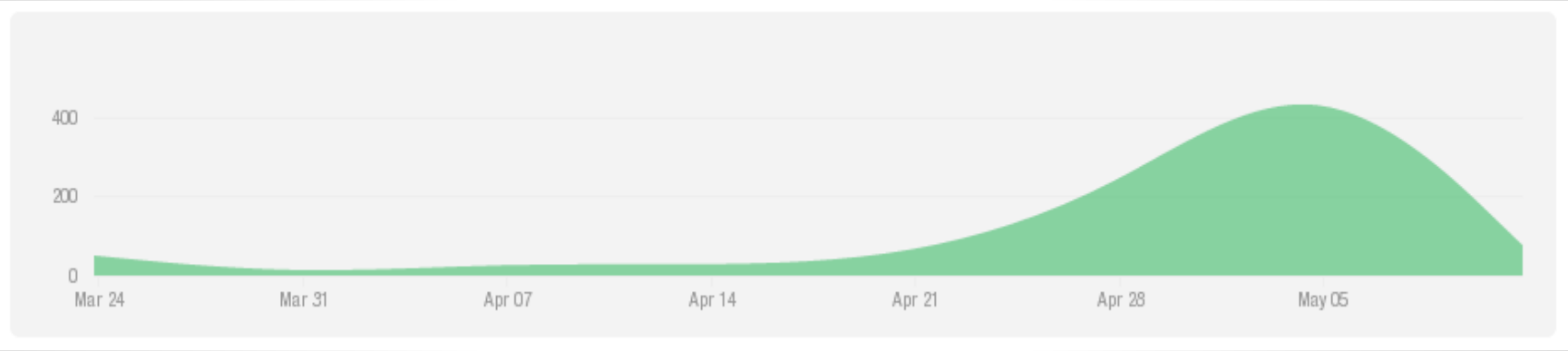
- Iterative and incremental development



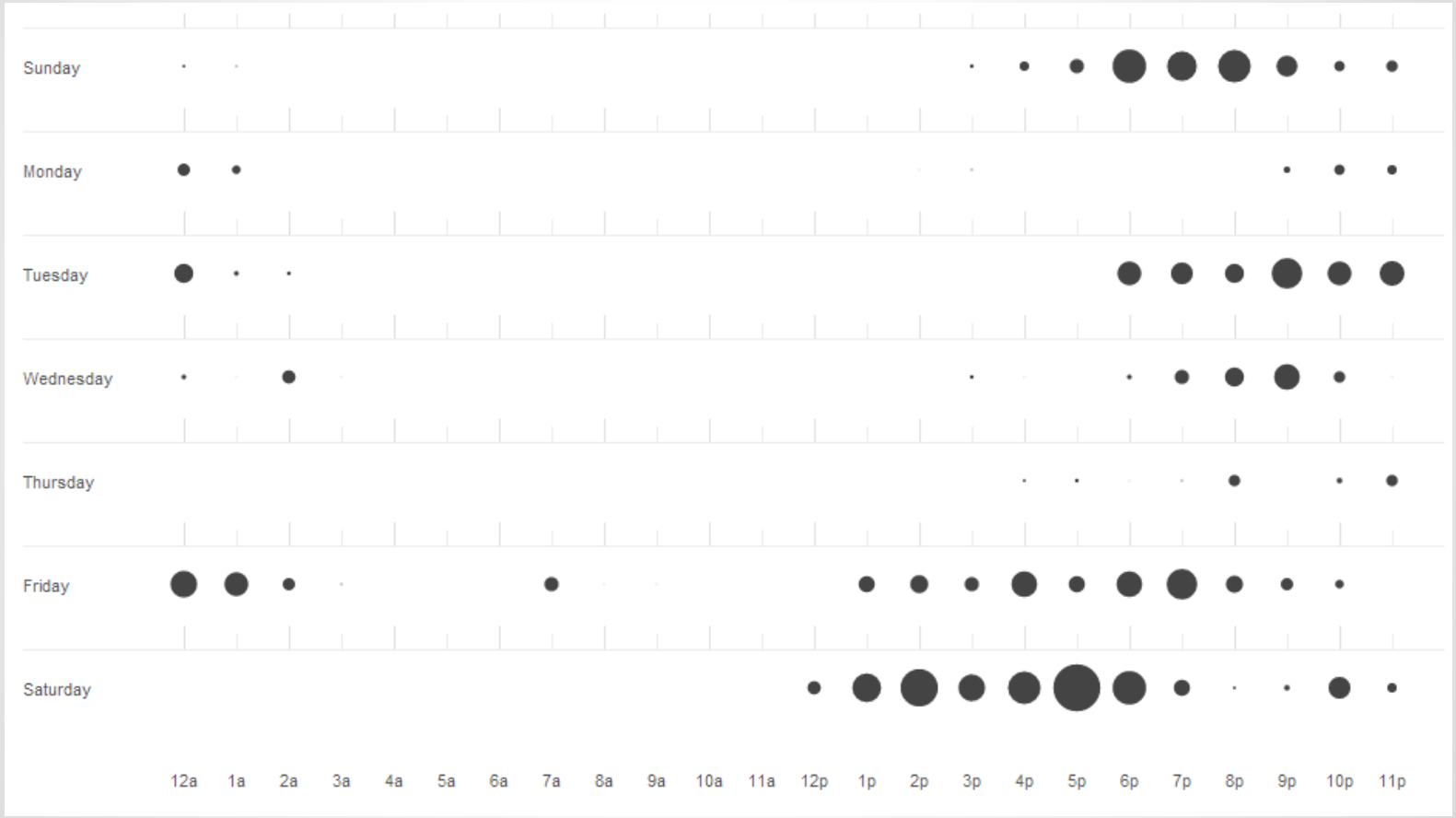
- First meetings: build grammar
- Later: group time for design and testing, busywork done individually

PROJECT MANAGEMENT

Volume of GitHub commits by date



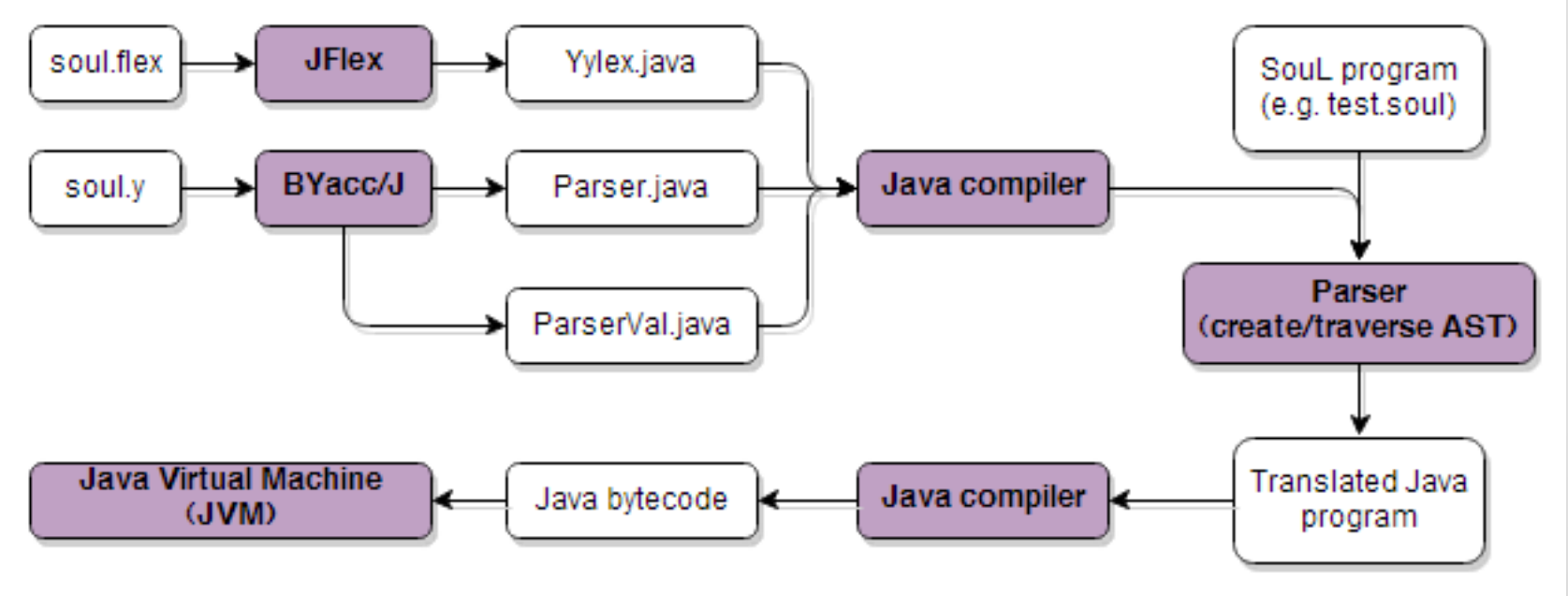
PROJECT MANAGEMENT



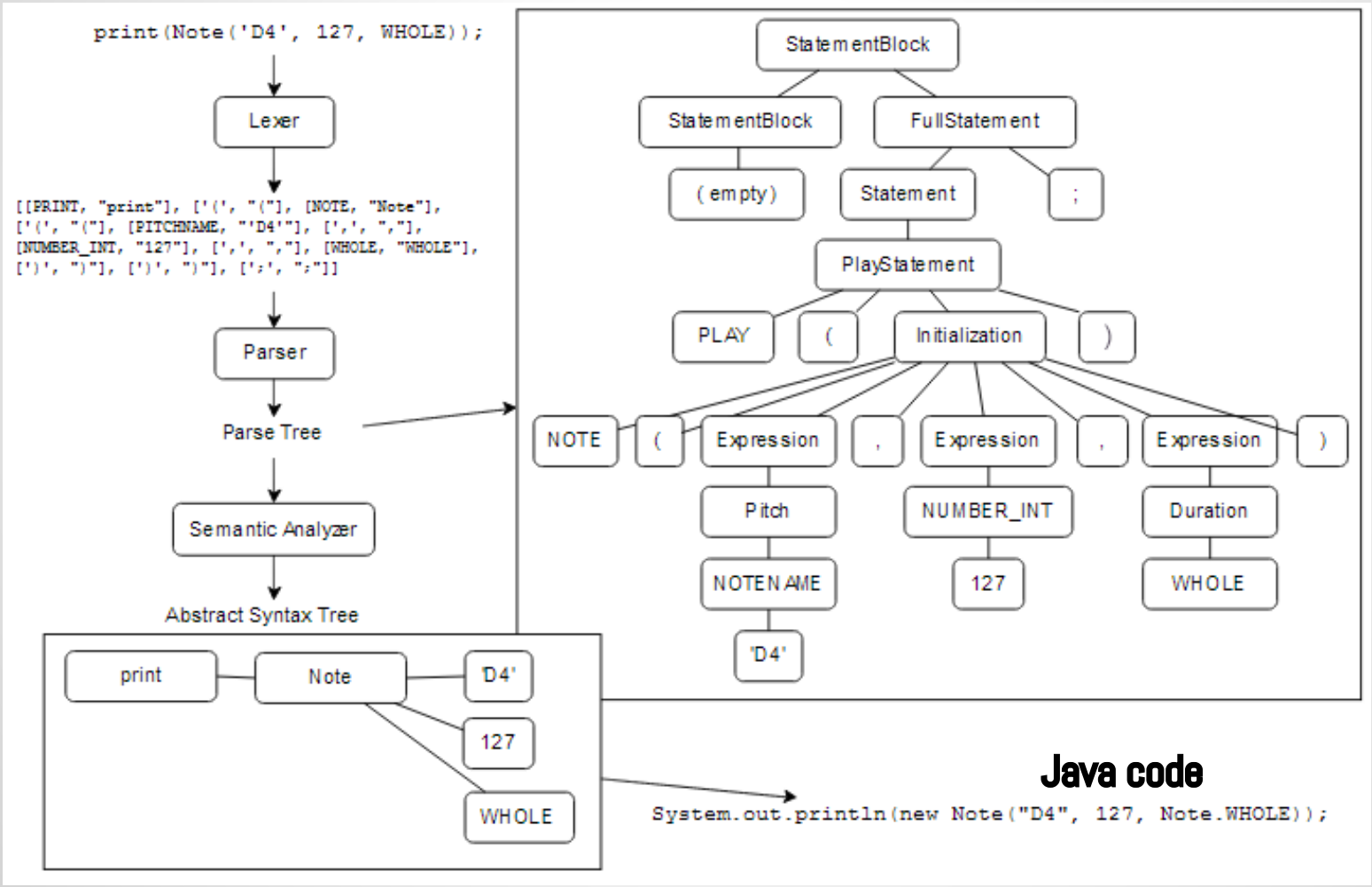
SouL
SOUND LANGUAGE

TRANSLATOR ARCHITECTURE

Block diagram of the SouL compiler:



TRANSLATOR ARCHITECTURE



THE SOFTWARE WE USED

GitHub

SouL
SOUND LANGUAGE

RUNTIME ENVIRONMENT

- Works on any UNIX system with proper tools installed
- Execution
  - ./soul filename.soul
    - Prints all errors to user - both compile-time (type-checking) and runtime errors (divide by 0)
  - ./soul twinkle.soul - constructs an AST, translated into Soul.java, compiled into Soul.class with javac, and run with Java call

SouL
SOUND LANGUAGE

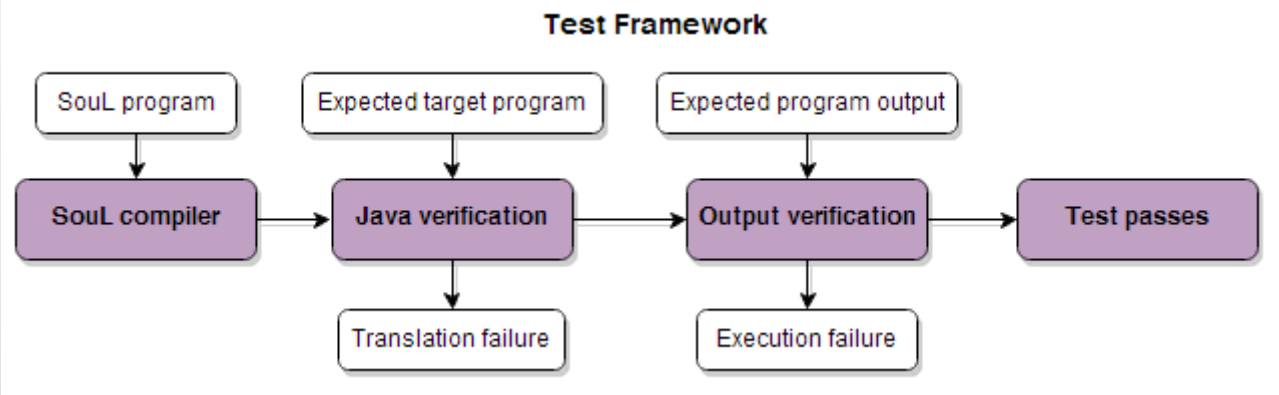
COMPILER-GENERATOR TOOLS

- **JFlex** - implementation of lex for Java
  - **jflex lexer.flex**
    - **Yylex.java**
- **BYacc/J** - implementation of yacc for Java
  - **yacc -J grammar.y**
    - **Parser.java**
    - **ParserVal.java**
- **Makefile**
  - make clean
  - make

SouL
SOUND LANGUAGE

TESTING

- **test_suite.sh**: Two tests
  - Same translated Java code
  - Same output
    - Output for playing not possible, so user has to check that the notes are the same as expected when running the test suite.



FUTURE UPDATES

- Extracting tracks from Sequence objects
- Overlapping Note and Chord objects at different ticks
- More complex data structures
  - Lists, Arrays, Hashtables
- User-defined functions
- Simplify syntax to make SouL less Java-like

SouL
SOUND LANGUAGE

CONCLUSION

- What worked well
  - Version Control
  - jsoul
- What we would've done differently
  - More regular meetings with mentors
  - Start with Java instead of lex/yacc
  - Schedule weekly goals and deadlines more often
- Lessons learned
  - Start early!
  - Research tools thoroughly before implementation
  - Create regression tests from the beginning
- Why to use SouL
  - Simple way to programmatically write music
  - Very few lines of SouL translate into many lines of Java

SouL
SOUND LANGUAGE

THANK YOU!

```
play("NeverGonnaGiveYouUp.mid");
```

SouL
SOUND LANGUAGE