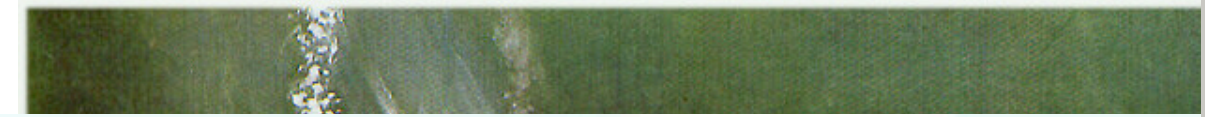
Engineering Iron Man's JARVIS

Dr. Emily Hill

Drew University

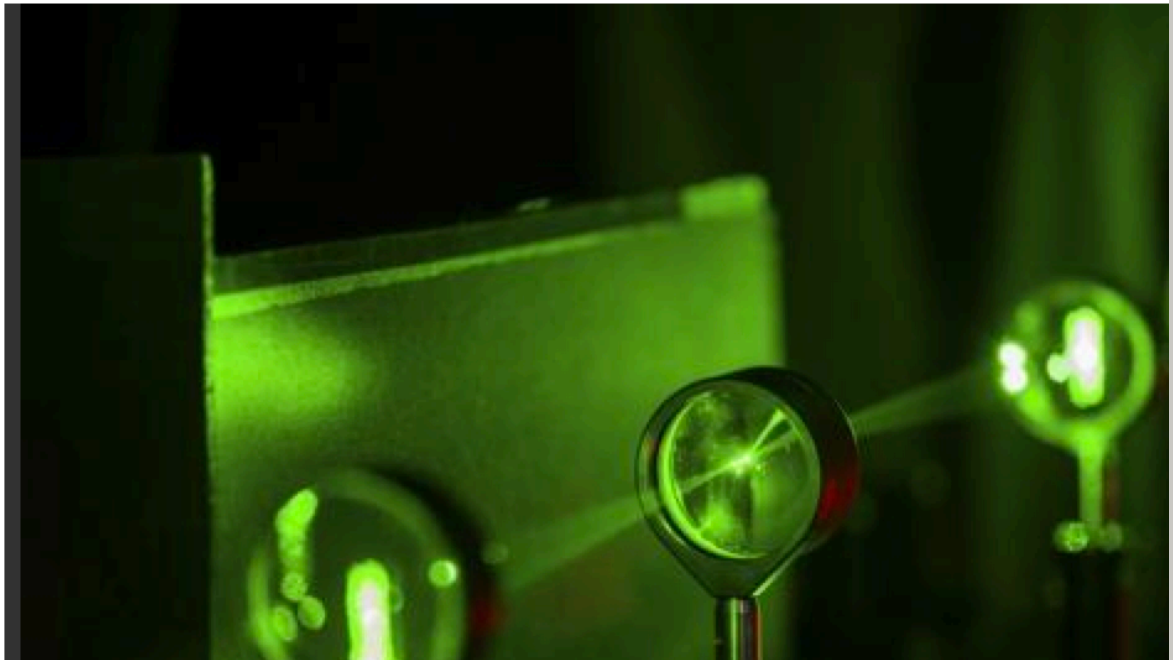
OOPSLE 2015

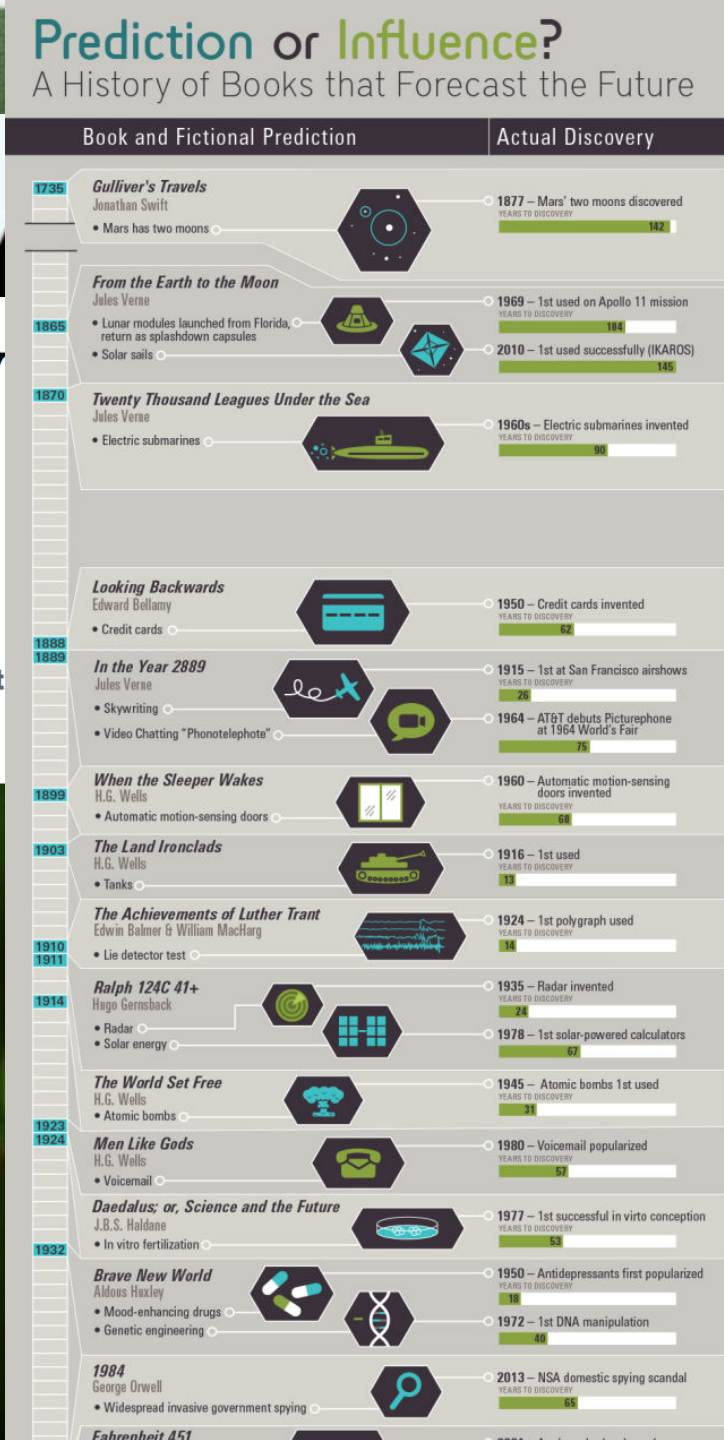


New York scientists unveil 'inv cloak' to rival Harry Potter's

BY CAURIE PUTNAM

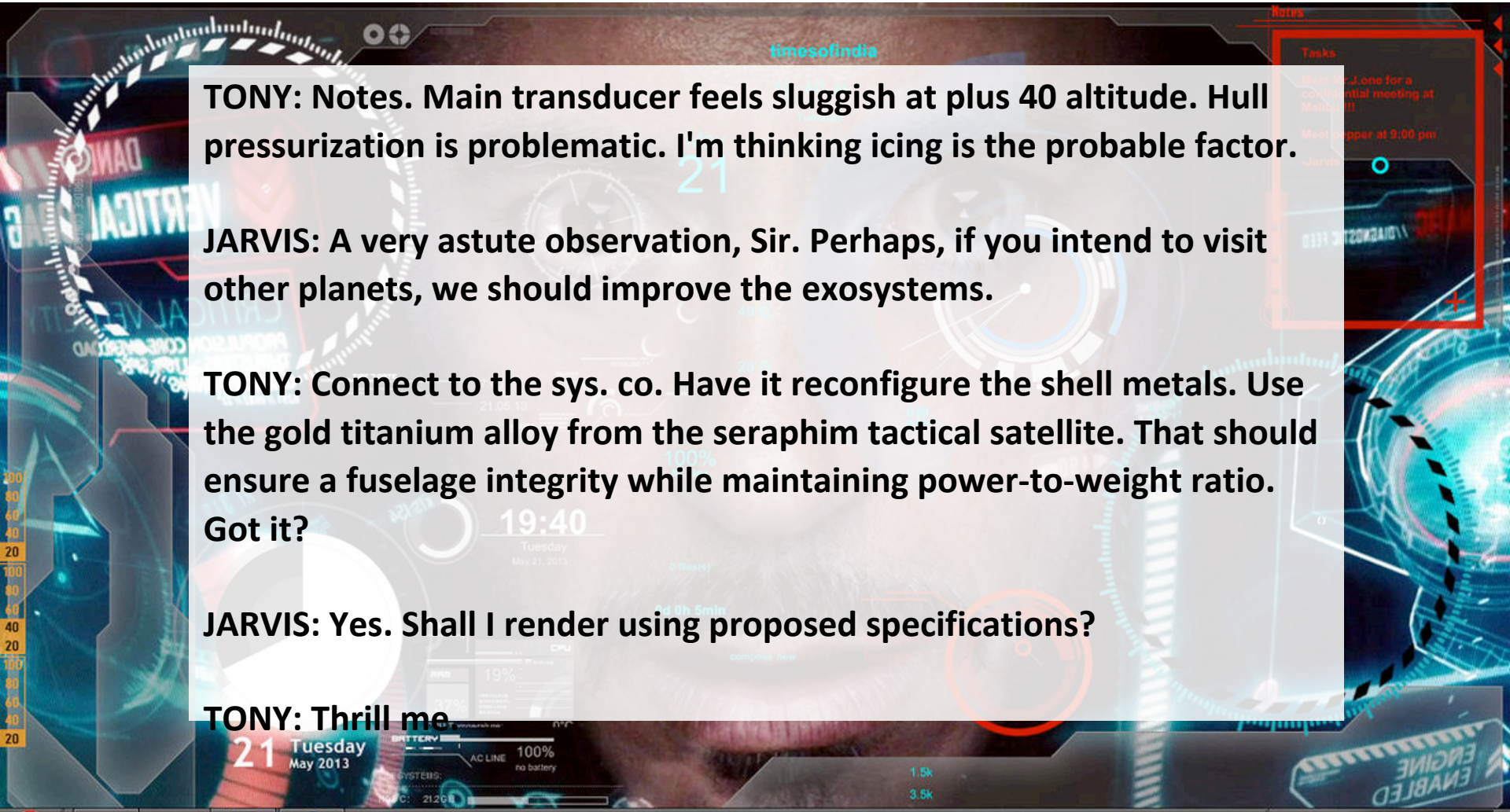
ROCHESTER N.Y. | Fri Sep 26, 2014 9:43pm EDT





JARVIS:

Just A Rather Very Intelligent System



TONY: Notes. Main transducer feels sluggish at plus 40 altitude. Hull pressurization is problematic. I'm thinking icing is the probable factor.

JARVIS: A very astute observation, Sir. Perhaps, if you intend to visit other planets, we should improve the exosystems.

TONY: Connect to the sys. co. Have it reconfigure the shell metals. Use the gold titanium alloy from the seraphim tactical satellite. That should ensure a fuselage integrity while maintaining power-to-weight ratio. Got it?

JARVIS: Yes. Shall I render using proposed specifications?

TONY: Thrill me.

Programming Jarvis

- Connect to the sys. co. Have it reconfigure the shell metals. Use the gold titanium alloy from the seraphim tactical satellite. That should ensure a fuselage integrity while maintaining power-to-weight ratio. Got it?
- Commands:
  - Connect
  - Reconfigure with parameters (use) & constraints (ensure…while)
  - Confirmation

Other applications?

- Anecdotally, hardest challenge for novice programmers is translating high-level NL description of problem into 7 basic programming concepts (variables, lists, loops, functions, conditions, etc)
- Could we meet the coming shortage of programmers by making programming languages operate at a higher level?
- Or at the very least, make better programming tutors?

How close are we?

Code Search

Code Completion

Keyword Programming [Little ASE '07]

Naturalistic Programming Languages [Knoll & Mezini OOPSLA '06

People-Specific Languages [Poss OOPSLE '14]

Method invocation search

Informal Software Representations [Arnold & Lieberman OOPSLA '10]

Representing NL Phrasal Concepts [Hill 2010]

How close are we?

- Keyword Programming (Little ASE 2007)
- Code completion
- People-specific languages (Poss OOPSLE 2014)
- NLP (Hill 2010)

Naturalistic PLs

(Knoll & Mezini OOPSLA '06)

Pegasus reads ideas described in a natural language, the *input program*, and to creates from it a fully executable program file, the *output program*, realizing the described ideas. For instance, Pegasus could read this natural language description:

```
Write three times: "I can understand you!".
```

and generate the following Java program, realizing the idea of the natural language d

```
class Main
{
    public static
    {
       for (int i
          System.o
    }
}
```

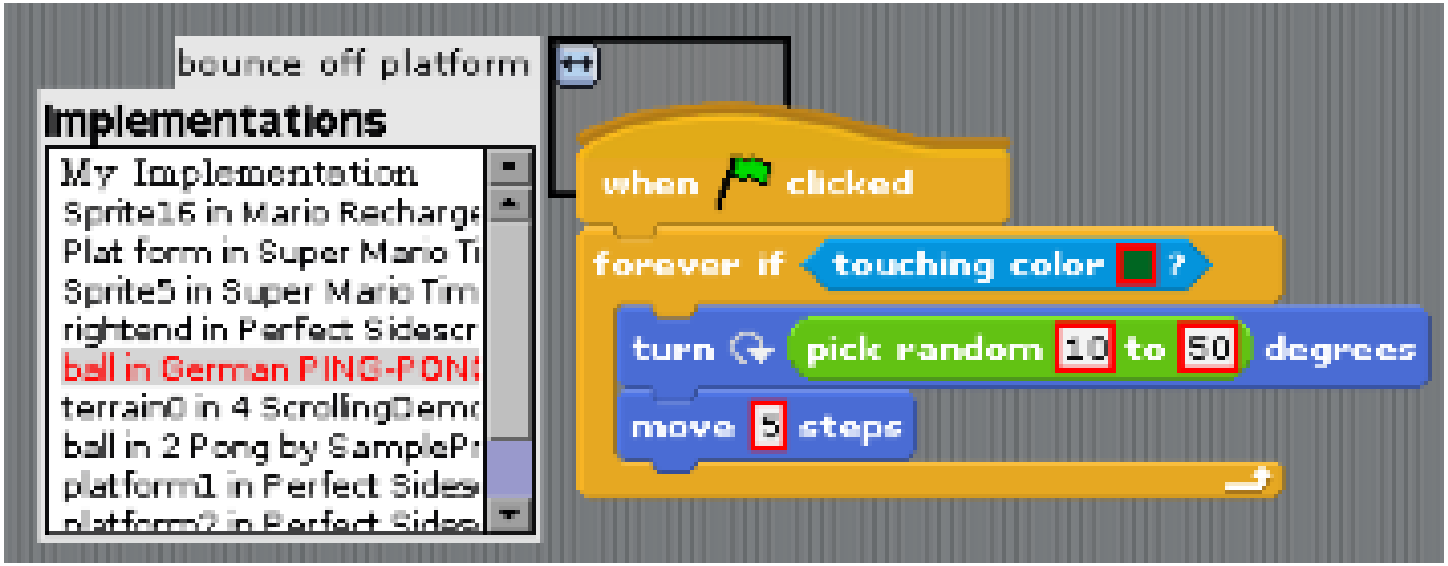
Challenges:

- **unnaturally precise wording?**
- **Dealing with ambiguity & domain knowledge**

People-Specific Languages

- Domain-specific languages taken to the extreme
- Open problems:
  - What personality/programming traits should be part of a PSL?
    - Male/female?
    - Personality features: intro vs extrovert?
  - Tool support to convert between PSLs

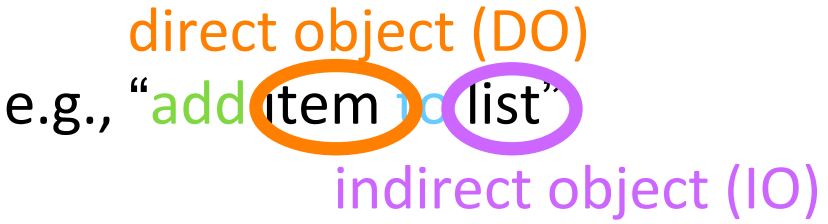
Informal Software Representations



(b) Given a purpose statement, the Zone sidebar (left) shows code that might fulfill it. Selecting an implementation from the list on the left shows its code on the right. As a simulation of future functionality, red boxes surround values that vary among otherwise similar code, highlighting what might need to be changed.

Phrasal Concepts in Source Code

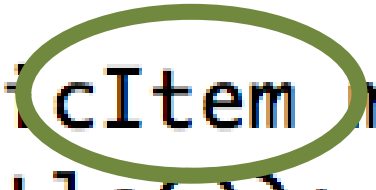
- Phrasal concepts generalize to arbitrary phrases using 4 types of semantic roles:
  - action (verb)
  - theme (direct object)
  - secondary argument (preposition + indirect object)
  - auxiliary arguments (any remaining signature information)



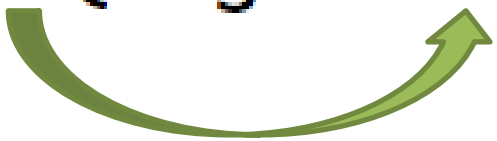
Method	action	theme (secondary arg)
`SetGroupsTest.tearDown()`	tear down	set groups test
`Restriction.convertToMinCardinality(int)`	convert	restriction (to min cardinality)
`addEntry(AuctionEntry ae)`	add	entry auction entry

Phrasal Concepts in Source Code

```
public void addMusic(MusicItem m) {
```



```
public void loadMusic(MusicItem m) {
    musicList.add(m.getTitle());
}
```



Method invocation sequences: RANDOOP

- Input:
  - classes under test
  - time limit
  - set of contracts
    - Method contracts (e.g. “o.hashCode() throws no exception”)
    - Object invariants (e.g. “o.equals(o) == true”)
- Output: contract-violating test cases. Example:

no contracts violated up to last method call

```
HashMap h = new HashMap();
Collection c = h.values();
Object[] a = c.toArray();
LinkedList l = new LinkedList();
l.addFirst(a);
TreeSet t = new TreeSet(l);
Set u = Collections.unmodifiableSet(t);
assertTrue(u.equals(u));
```

fails when executed

Problem Summary: Translation

- High-level NL description of problem ➡ executable code
- Possible solutions/directions:
  - Pull out actionable VPs from NL & search for code examples or chains of invocations (a la RANDOOP)
  - Constrain NL (a la Pegasus) or train PL with people-specific DSLs
  - Learn correspondence between HL NL -> PL
    - Documentation-code mining (naming conventions & naming bugs)
    - Apply comment generation rules in reverse

End-user naturalistic programming:

Feasible? Pipe dream?

Why so little progress
from PL community?