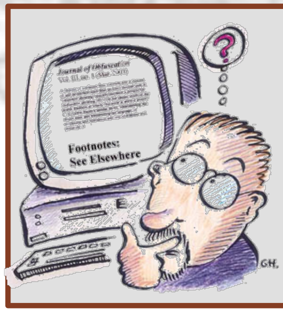
The Case for Software Evolution

Claire Le Goues*, Stephanie Forrest+, Wes Weimer*

*University of Virginia +University of New Mexico

- Maintenance = up to 90% of a project's cost, up to $60 billion in the US annually.

The current software development paradigm is broken.



- Software has become too complicated for humans to understand.

- Most aspects of a software system change over its lifetime.

We should treat software like a complex, evolving system.

- Human modifications resemble evolutionary mechanisms.

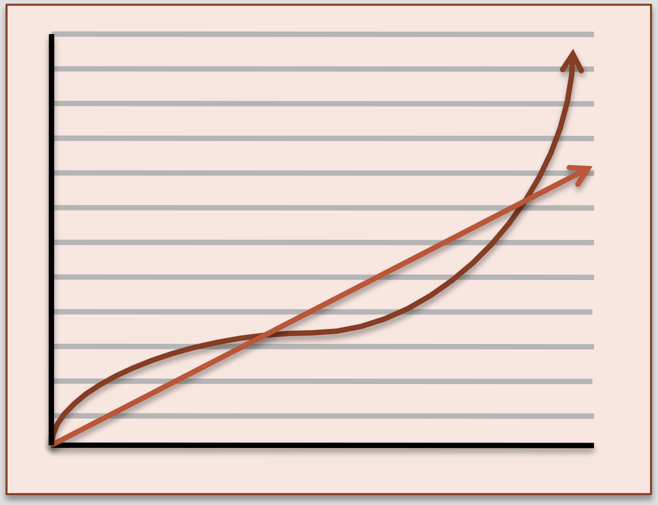
This perspective challenges several current research assumptions.

1. Soundness

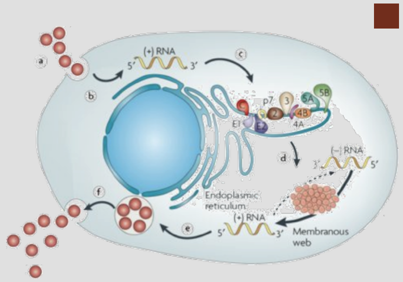
- Complexity limits the feasibility, utility of precise proofs of program properties.
  - Biological systems do not rely on *a priori* correctness.
- **Future directions:** new definitions of utility; program analysis features that enable practical adaptation.

2. Definition of acceptability

- Without soundness, we need new program analysis metrics and benchmarks.
- **Future directions:** test suites (evolving), continued execution, heuristics.
  - Test case generation that produces full test cases, with expected output.



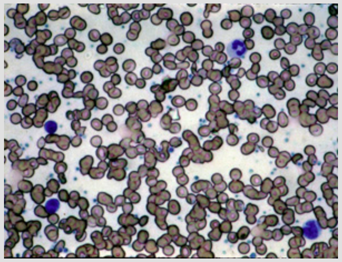
3. Separation of concerns



- Biological boundaries enforcing modularity are much richer than their computing equivalents.
- **Future directions:** relax hardware/software abstraction to achieve robustness in dynamic and energy-constrained environments.

4. Homogeneity

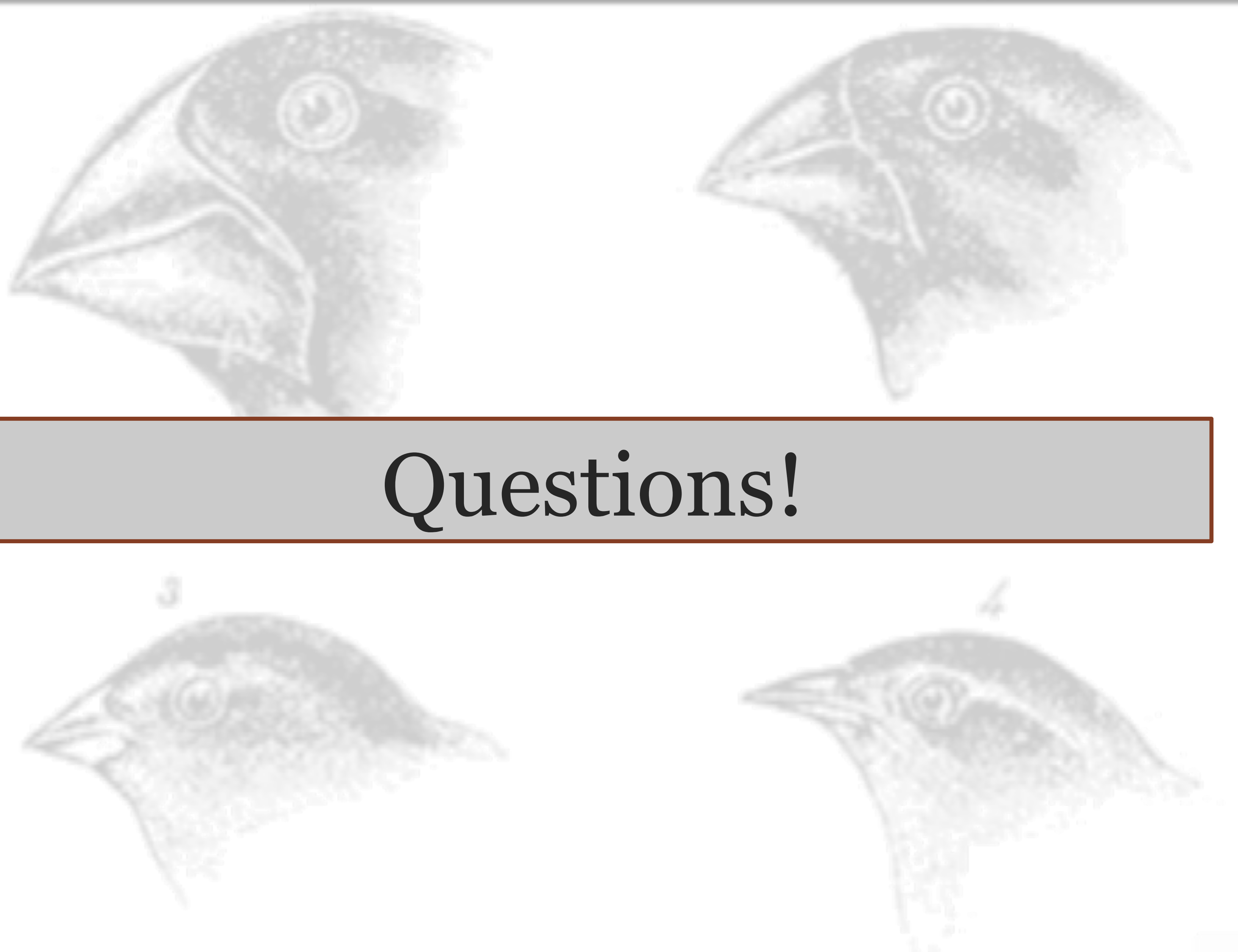
- Biological diversity is an important source of robustness.



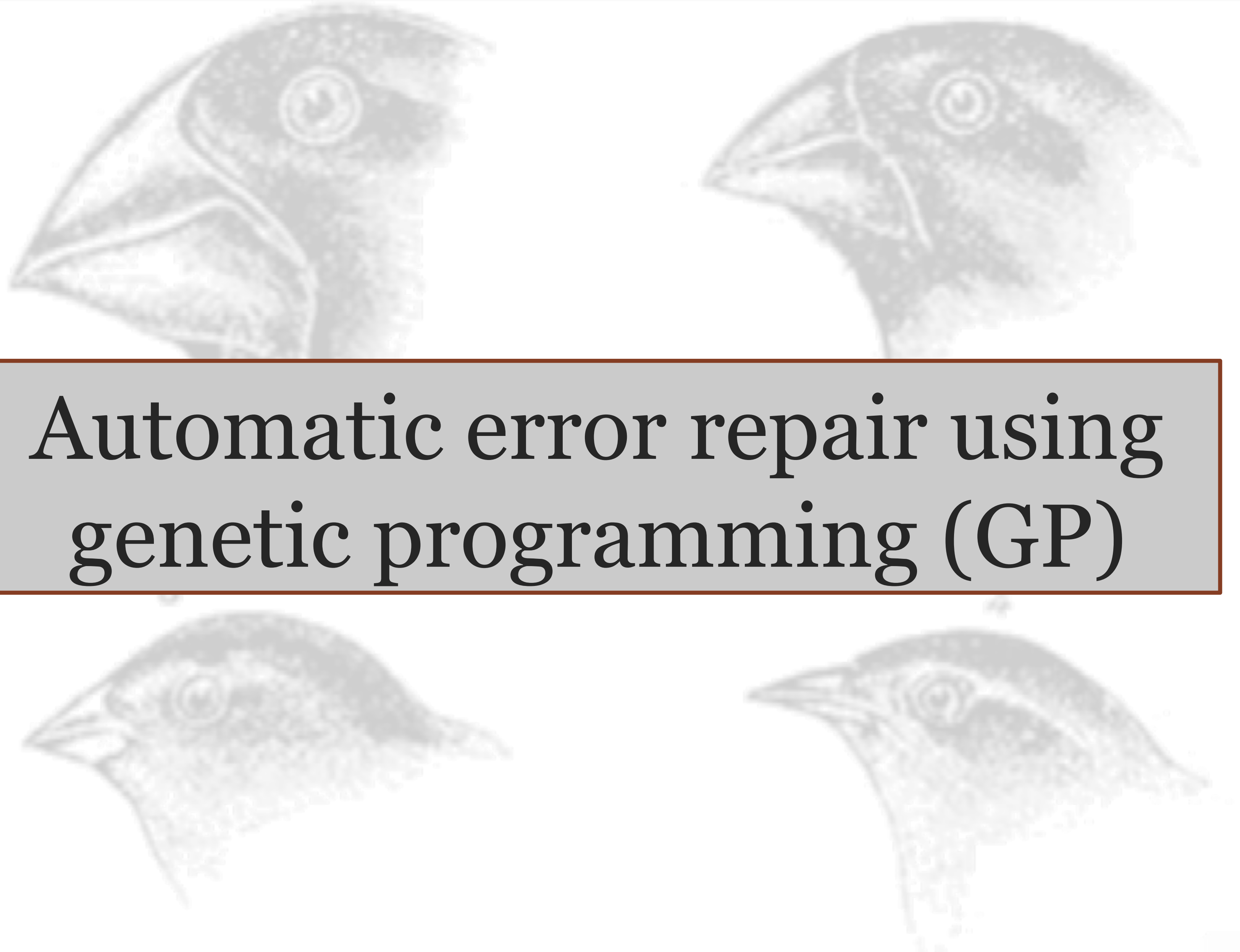
  - Protects against the spread of disease.
  - Provides alternative pathways to maintain functionality.
- **Future directions:** research techniques that account for and leverage diversity.

Conclusions

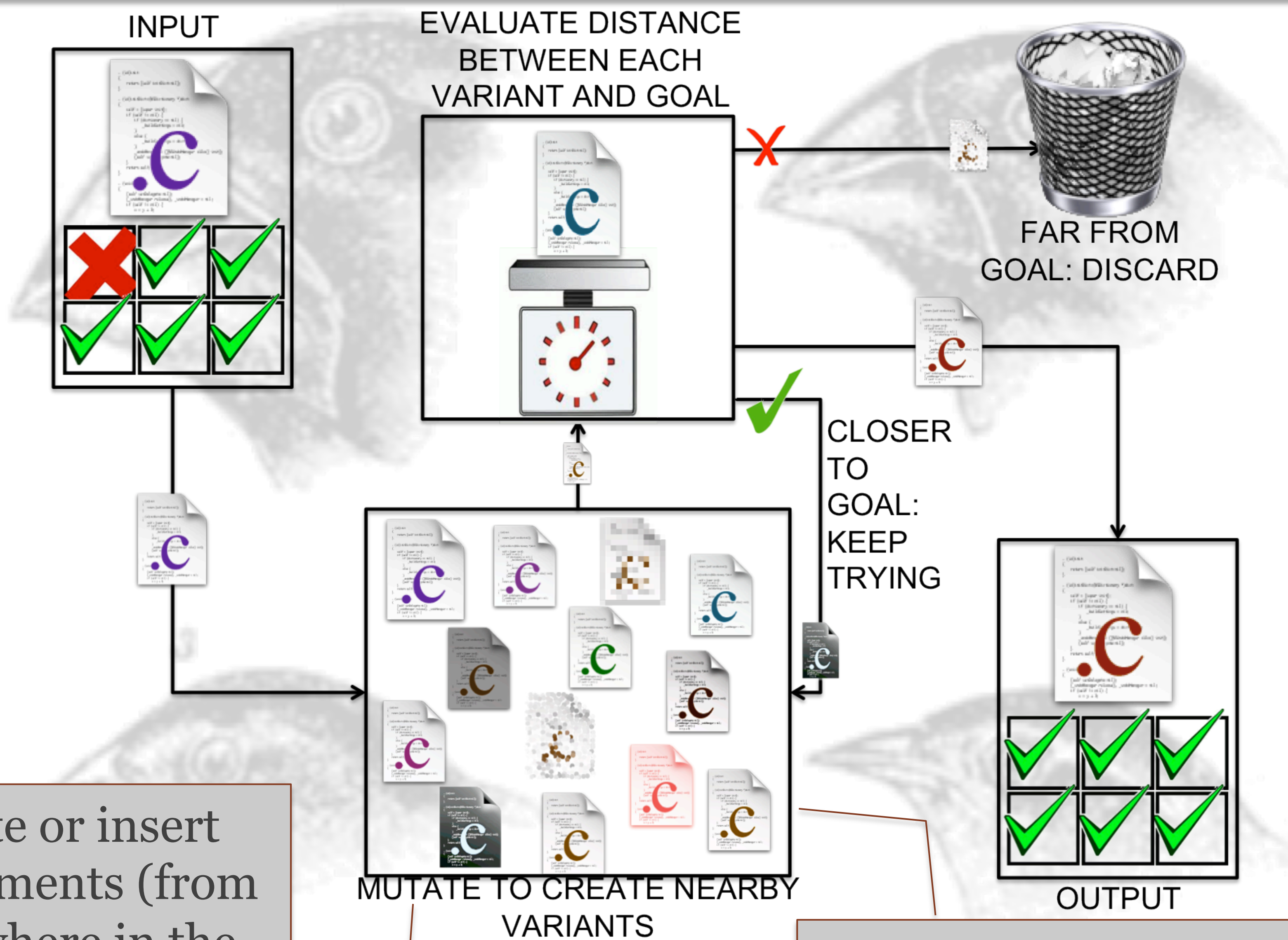
- We should think of computational systems as complex evolving systems.
- This could dramatically change software development and maintenance.
  - May be able to revisit the dream of automatic programming.
  - May enable theoretical analyses of how software is likely to operate over long time scales.



Questions!



Automatic error repair using genetic programming (GP)



Delete or insert statements (from elsewhere in the program).

More likely to change fault-localized regions.

- **Results:** repaired 15 legacy C programs (> two million LOC); < 5 minutes (average); error types: buffer overruns, denial of service, format string vulnerabilities, infinite loops…
- Highlights analogy between software and complex evolving systems.
  - Assumes redundancy of functionality even in software executing in isolation.
  - Many bugs repaired by copying code between locations, resembling biological evolution.