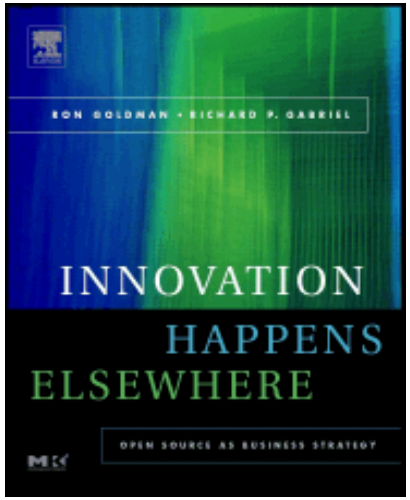
# New Approaches to Computing: Learning from Biology

Ron Goldman
Sun Labs

# Who am I?

- Researcher at Sun Labs
- Software developer
- Open source expert
  - Author: (with Richard P. Gabriel) *Innovation Happens Elsewhere: Open Source as Business Strategy*



# Acknowledgments

Ideas in this talk come from many people

- *Conscientious Software* co-author Richard P. Gabriel
- Santa Fe Institute
- Humberto Maturana and Francisco Varela

Thanks to Sun Microsystems for allowing me to research Biologically-Inspired Computing

# Talk outline

- Trends in computing
  - We are in big trouble
- Maybe ideas from biology can help us
  - Some already have
- A possible direction
  - Conscientious software

# Trends in computing

- Increasing program size
- Increasing program duration
- Increasing complexity

# Fifty years ago

- Early software applications were small
- Were self-contained
- Ran for a limited time
- Ran on a single computer
- Linear execution

# Computer Science

- Strong Mathematical influence
- Analysis of algorithms
- Idea that program could be proven correct
- Possibility of bug-free code

# Current software

- Applications now large and getting larger
- Applications interact with other apps
- Run continuously
- Run distributed over many computers
- Parallel execution

# Modern software engineering

- Waterfall => iterative design
- Describe application via specification
- More pragmatic focus on shipping systems
- Try to minimize bugs through testing

# Future software systems

- Ultra-Large-Scale (ULS) systems
- How can we build systems of the future that are likely to have billions of lines of code?
- Run on thousands of computers
- Developed by different groups
- Continuous operation, continuous development/evolution
- Needs to monitor and repair itself

# Effects of increasing complexity

- More bugs
- More brittleness
- More susceptible to attack

# Famous software engineering disasters

- North America blackout (2003)
- USS Yorktown (1998)
  - a mistakenly entered zero data value caused a cascade of errors that eventually shut down the ship's propulsion system
- Denver airport baggage system (1994)
- AT&T Network Outage (1990)

And many other failed systems that have been cancelled or were never delivered

# We can't get there from here...

# Why look to biology?

- Biological systems exhibit many properties we wish our computer systems had
  - Robustness
  - Decentralized
  - Adaptive
  - Efficient
  - Beautiful
- In 4 billion years Nature has evolved many solutions to tough problems

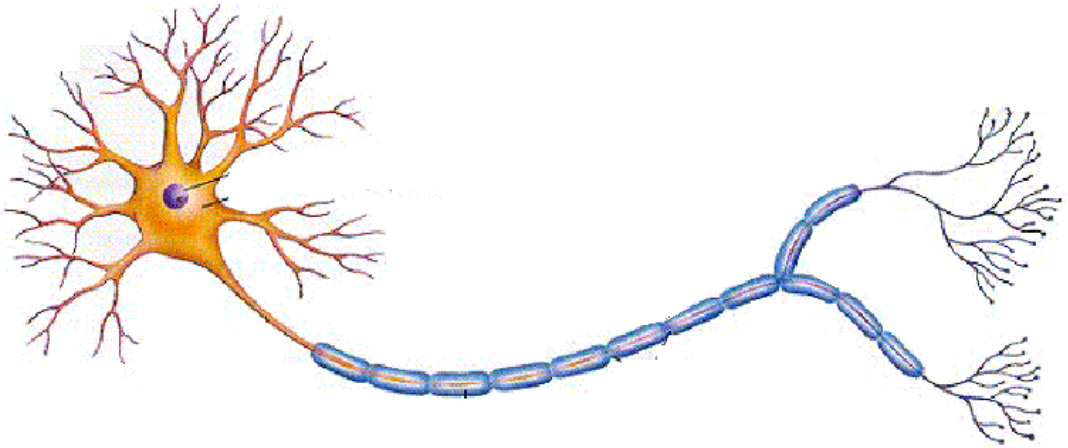
# What have we already learned from biology?

- Neural nets
- Genetic algorithms
- Swarm intelligence
- Immune system

Note: Not the first time computing has turned to biology

- Biological Computer Laboratory, 1958–1976
  Heinz von Foerster

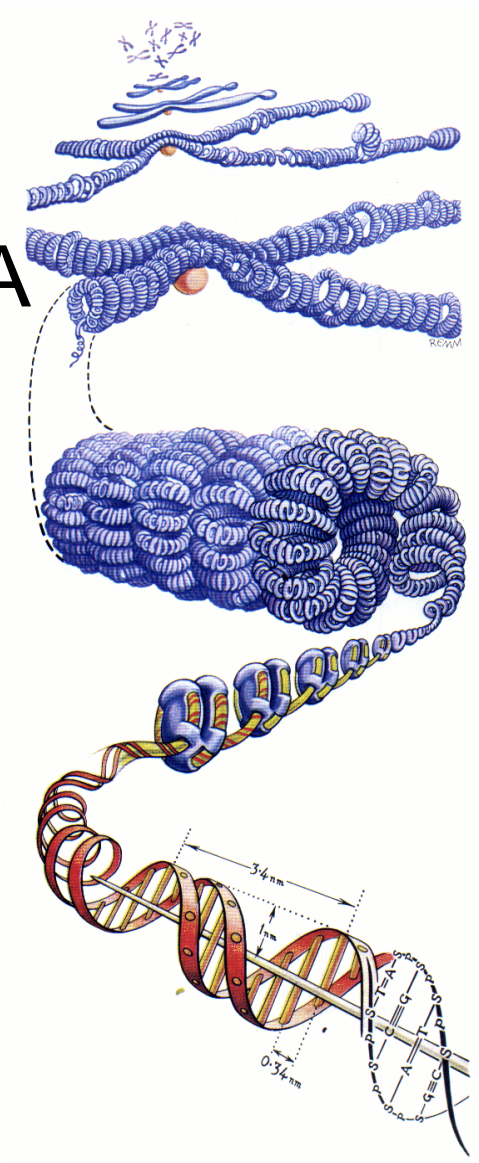
# Neural Nets



- Looking to the brain as a model of computation
- Idealized neurons connected together
- Inspiration for von Neumann & onwards
- Applications include
  - Recognition of handwriting, speech, OCR
  - Financial classification: credit card fraud, loan approval

# Genetic Algorithms

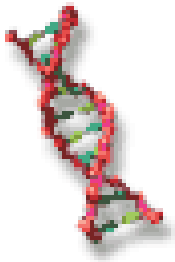
- Evolution via modifications to DNA
  - Inheritance
  - Mutation
  - Recombination
  - Natural selection
- Can rapidly locate *good* solutions, even for difficult search spaces
- John Holland pioneered field



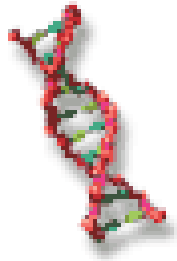
# Genetic Algorithms: Job Scheduling

John Deere seed planter assembly line

- Over 1.6 million different product configurations
- Originally scheduled manually
  - Factory throughput was not good
- Introduced GA to breed a good schedule
  - 40,000 generations bred each night
  - Best one used next day
  - Has been highly successful



# Genetic Programming



- Applying genetic algorithms to programming
- John R. Koza pioneer in field
  - 36 instances where genetic programming has produced a human- competitive result
  - Experiments using a 1,000-node (Pentium II) Beowulf-style parallel cluster computer

# Swarm Intelligence

- Social insects (ants, bees, termites) are highly successful
  - Flexibly adapt to a changing environment
  - Robust even when one or more individuals fail
  - Self-organize without needing centralized control
- Emergent behavior
  - Local rules yield global behavior
  - Simple rules can produce complex collective behavior

# Swarm Intelligence: Applications

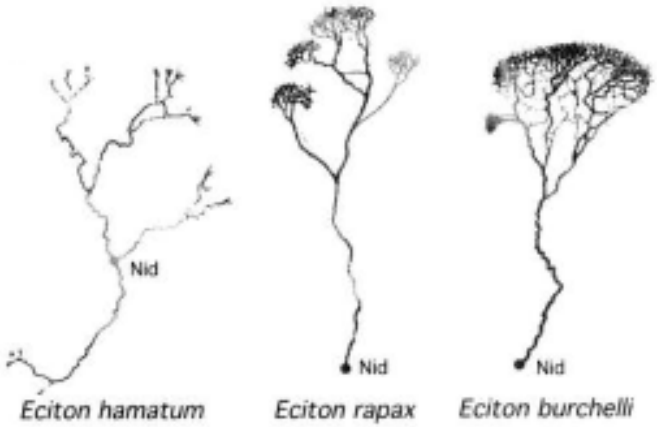
Truck painting at GM

- Based on how honeybees shift work roles
- Changing paint color is time consuming & costly
- Auction system: booths bid based on current color & queue
- Saved over $1 million/year just in paint
- Handled breakdowns and surges in orders flexibly and robustly

# Swarm Intelligence: Applications

Phone & Internet routing

- Based on ant foraging
- "Digital pheromone" used to reinforce paths through uncongested areas




- Being explored by France Télécom, British Telecom & MCI WorldCom

- Simulations suggest better than current Internet protocol (Open Shortest Path First)

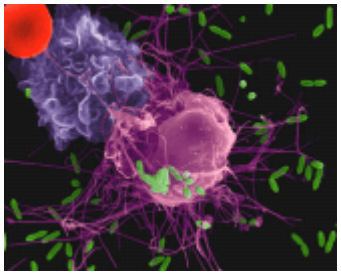
# Emergent Behavior: Issues

- Decentralized, hence scalable
- Difficult to specify the local rules
- Difficult to prove correctness
- No “guarantee” of result
- Many people not comfortable with it

---

- Need new tools to link the local to the global
- New ways to build and think about such systems
- Need better understanding of resilience and stability properties

# Immune System



- Human immune system provides multiple layers of protection
  - Innate: macrophages, complement system, …
  - Adaptive: antibodies, T cells, …
- Can respond quickly to previously encountered organisms
- Can adapt to new infectious organisms it has never seen before
- Immune system needs to distinguish between self and other

# Immune System: Applications

- Stephanie Forrest, University of New Mexico
  - Use an application's sequence of system calls or network messages to describe application
  - Train system for normal behavior (identify self)
  - Create detectors that do not match self to identify possible infections
- Steve Hofmeyr, Sana Security
  - *Primary Response* product aimed at protecting network servers
  - Learns normal behavior of server programs
  - Recognizes attacks and blocks them or alerts sysadmin

# Digging deeper

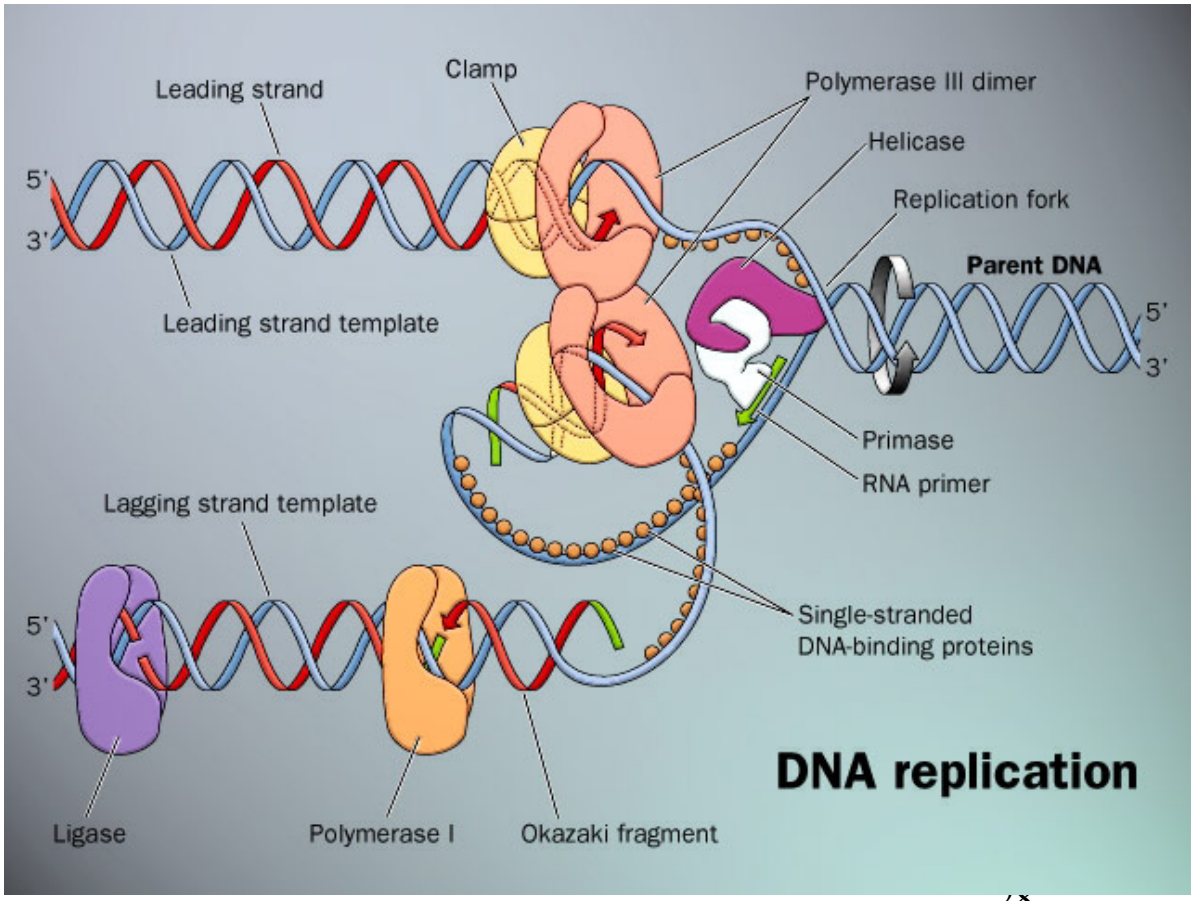
- How can we apply ideas from biology to building robust, distributed computer systems?
- How will they change our current practices?
- How will they change how we think about building systems?

# Robustness

- How does biology handle critical tasks
  - Cell reproduction

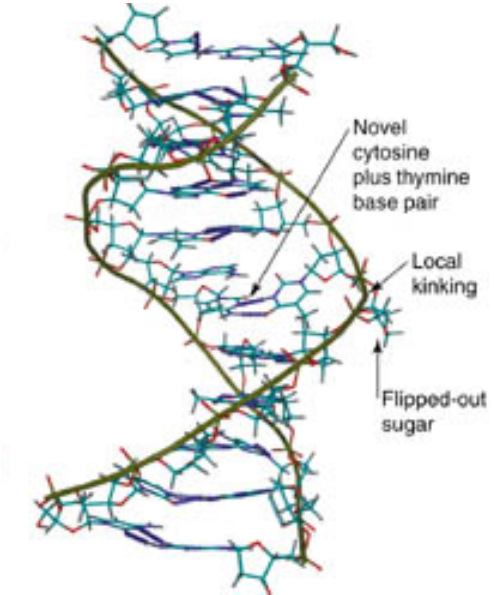
# DNA replication

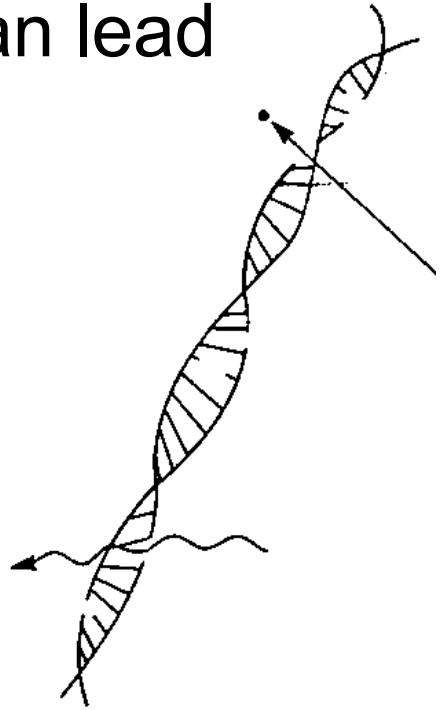
- Accurate copying is essential to reproduction
- E. coli has about 30 genes involved in copying DNA



# DNA damage: kinks, breaks, gaps, mismatches, …

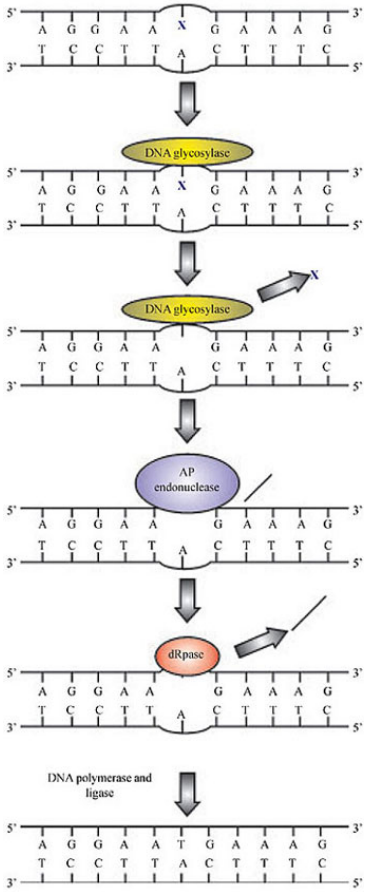
- Very common:
  - 10,000 times per day in every cell a stretch of DNA will lose one of its constituent base
  - If unrepaired, the damage can lead to disease, notably cancer.

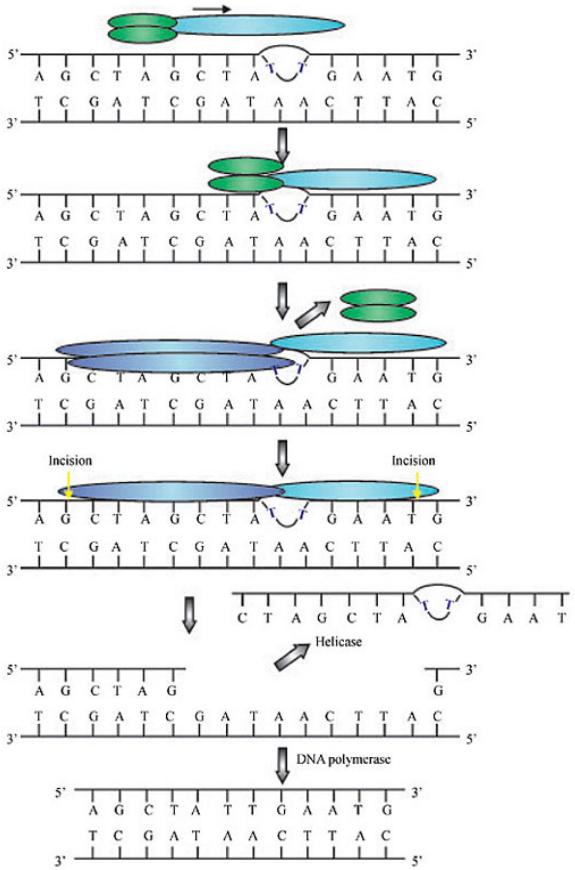


# DNA repair

- E. coli has 63 genes to do DNA repair



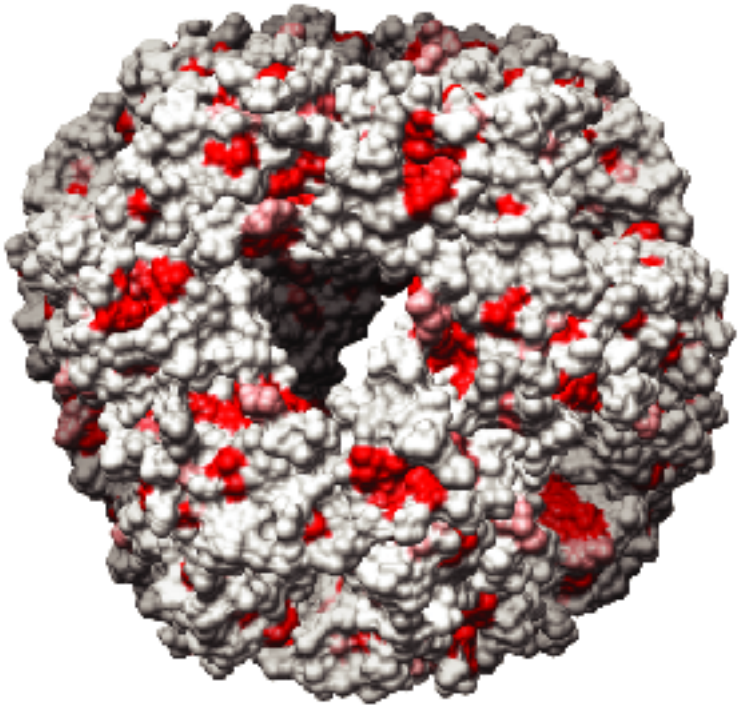



# DNA copying vs. repair

- E. coli: copy (30) vs. repair (63)
- M. genitalium: copy (25) vs. repair (13)
- Replication happens only during cell division
- Repair happens continuously via many processes
- Also happens during replication & transcription

# Other repair mechanisms

- Heat shock chaperone protein
  - Heat causes proteins to lose their proper shape
  - Need to refold them

# Errors are normal

- Biology assumes that errors will always occur
  - Externally: unexpected environmental conditions, attacks
  - Internally: process failures, data corruption
- Need to constantly monitor internal processes & external conditions
  - Monitoring involves multiple feedback loops

# Repair is essential

- Need to recover from errors once detected
  - Switch to alternative process
  - Repair (or kill) defective units
- Efficiency is important, but survival is the top priority

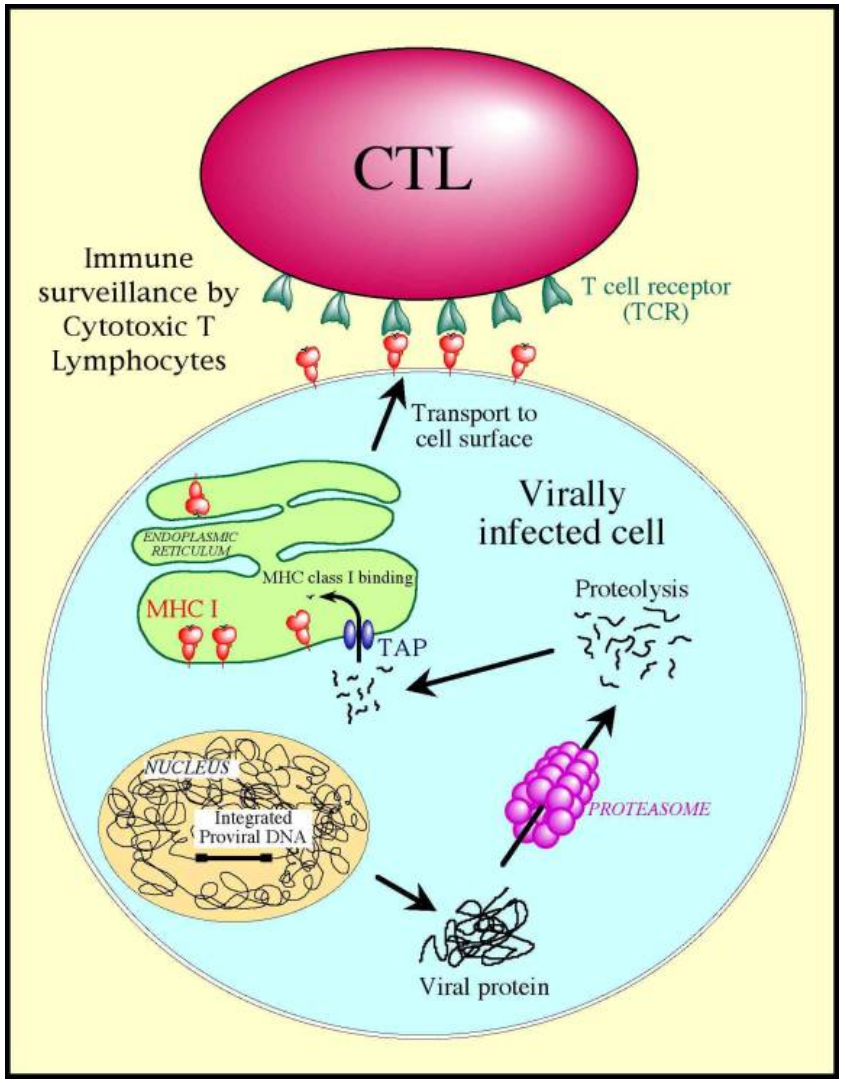
# Error detection and repair in computer systems

- Currently little or no error handling code
  - 5-10% typical
  - 5ESS software had ~33% error handling code
- No surprise that software is not robust
- Lesson from biology:
  - More resources devoted to repair than basic functionality
- Need to fundamentally change how we build software systems

# Visibility

Our immune system:

1. Proteins broken down in cell
2. Fragments get displayed on cell surface
3. Immune system inspects

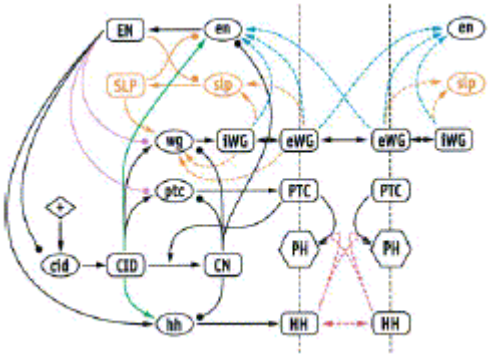


# Other Ideas from Biology

- Apoptosis (programmed cell death)
- Spatial compartmentalization
- Stigmergy (communication via changes to the environment)
- Symbiogenesis (symbiotic merging of independent organisms)
- Multiple nested feedback loops
- Developmental biology

# Complex Systems

- More than just a collection of interacting components & subcomponents
- Biological organism is a collection of mutually reinforcing feedback loops
  - Gene regulatory networks




- Interconnected feedback loops make for a more stable system, that is better able to adapt to internal or external changes
  - Dynamic stability

# Autopoiesis vs. Allopoiesis

- Autopoiesis: a system that (re)constructs itself
  - Example: living cells
- Allopoiesis: a system that makes an external product
  - Example: a factory

Our computer systems need both

- Current software is almost exclusively allopoietic
- How can we include autopoietic modules?

## Autopoietic principles

- visibility / transparency
- dynamic / flexible interfaces
- self-generating / decentralized
- diversity
- reactive

## Allopoietic principles

- information hiding
- static / rigid interfaces
- command & control / hierarchical
- uniformity
- manufactured

## Autopoietic principles

abundance / redundancy

loosely-coupled interaction

pattern-based

local rules

lithe languages

## Allopoietic principles

parsimony / efficiency

tightly-coupled interaction

signal-based

global reasoning

Java™, C++, C#, usual suspects

# Robustness and control

- Allopoiesis performs basic functionality
  - Current software does this
- Autopoiesis provides system robustness
  - Use of feedback mechanisms
  - Continuous testing & repair
  - Need new languages/constructs

# Conscientious software

- Software taking responsibility for itself
- Software as collaborator
- Adaptive installation
- Continual testing
- Continual noticing

# Software taking responsibility

- Software can act
- Sense some part of its environment
- React to changes
- Affect its environment

# Example: Word processing

- Automatically save all edits
  - Without making the person wait
- After a crash can restore previous state
- No work should ever be allowed to be lost

# Software as collaborator

- Examine its environment prior to installation
- Monitor changes to its operating conditions and adapt
- Pay attention to how it's being used and become easier to use
- Provide for its own improvement at the hands of local developers
- Accept and provide for its own death and replacement

# Adaptive installation

- More than bits properly stored
- Customizations
- Fitting in with environment / users
- Discovery versus preference setting
  - Prevailing color scheme
  - How is spellchecking done ’round here?
  - Emacs keybindings? How about Emacs itself for text editing

# Never-ending installation

- As other software is added
- As experiences of the users change
- Always revocable

# Complete Packaging

- Binaries
- Source code
- Build environment
- Tests

# Complete packaging

- Local adaptation
- No “lost source”
- Population of individuals
- Possibility of cross-breeding

# Continual testing

- Test environment before installation
- Test after installation
- Always (possible to be) testing
- User-written tests
- Send results to “original developers”

# Continual noticing

- Software / components running notice environmental conditions
- Adapt, adjust, feedback

# Visibility: software that mutters

- Internal vocabularies
- Continual updating / history
- Multidimensional (unlike telemetry)
- For adapting

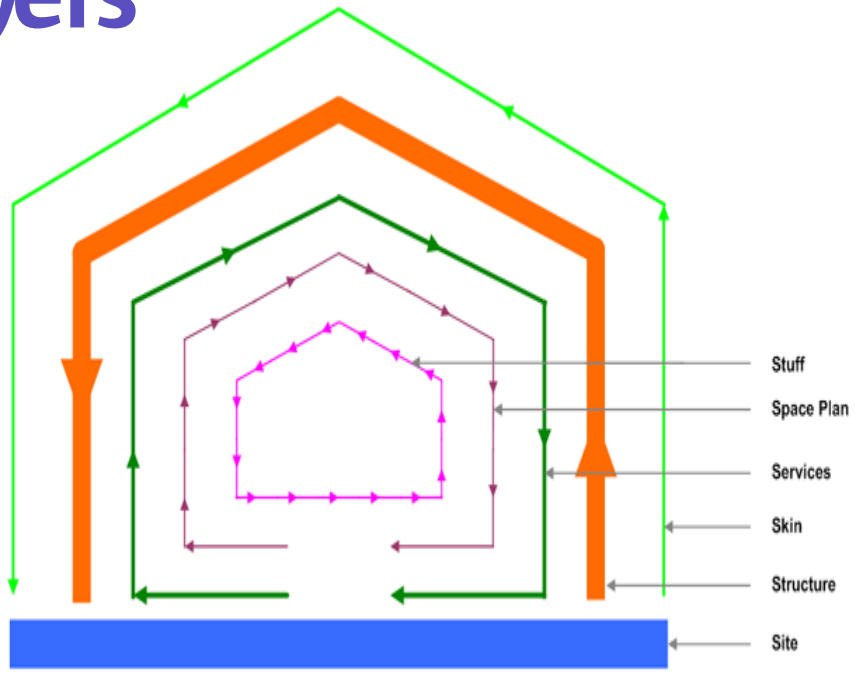
# Exercise

- Not just using
- Not just testing
- Improving / structural change through use
- Practice

# Design for continuous redesign

- Some properties are shared across all living creatures on the Earth
  - RNA three base code for amino acids
- These properties are highly conserved
- Other properties change rapidly
  - Immune system antibodies
  - Venom / anti-venom

# Stewart Brand's six layers




- Site: is eternal
- Structure: foundation and load-bearing elements
- Skin: exterior surfaces
- Services: wiring, plumbing, HVAC
- Space plan: walls, ceilings, floors, doors
- Stuff

# Signaling

- Direct/immediate communication
  - Nerve signals
  - Language
- Indirect/slow communication
  - Diffusion
  - Gradients

# Conscientious software principles

- Assume failure is common
- Seek out and repair errors
- Write tests and continually run them
- Use feedback
- Make things visible
- Flow
- Gradients
- Layers
- Diversity