# Reverse Engineering Case Study (Lecture 17)

## Analysis of Software Artifacts

# Reverse Engineering

- frequently, code over a long period of time is "worked on" by a lot of programmers
- different programmers
  - various coding styles
  - incomplete understanding about the program
  - local optimizations to code

# Reverse Engineering

- this leads to *code decay*
  - the structure of the program becomes very complex
  - usually because there isn't a consistent design philosophy
- usually happens to very intricate pieces of code
  - compiler optimizers notoriously suffer from the code decay disease

# The Reverse Engineering Process

- *Program understanding*
  - use existing documentation (if you are lucky!)
  - talk to the software architects
  - use program analysis tools
- *Model building*
  - build a model of what the system does using program understanding
- iterate between the two steps until the model stabilizes

# The Forward Engineering Process

- assume that the model has been constructed
- use the model to construct the code
  - use a modern programming language (`C, C++, JAVA`)
  - large effort going on in the industry to reverse engineer legacy `COBOL` code
- test the code and iterate the process
- frequently have to refine the model

# The CTAS Case Study

- case study was a one-semester graduate seminar at MIT
- twelve students and three faculty
- re-engineered a piece of the CTAS system called the *communication manager*

# What is CTAS?

- suite of tools to help controllers manage air traffic flow
- purpose is to increase landing rate through automated planning
- a prototype deployed at the Dallas/Fort Worth (DFW) terminal improved landing rate by 10 percent

# What is CTAS?

- receives input about
  - location, velocity, and flight plans
  - weather data and available runways
  - standard landing patterns and controller commands
- uses this data to suggest a landing sequence to minimize unused landing slots
- predicts aircraft trajectories as much as forty minutes in advance

# What is CM?

- The Communications Manager (CM) sits at the center
- acts as a message switch moving data among components
- maintains the database of aircraft information

# What is CM?

- basically is the communication hub for the entire CTAS system
- CM has become very complex
- repository for several unrelated features
- single point of failure in the entire system
- 80,000 lines of `C and C++` code

## What was done?

- used program understanding to build a model of the CM
- re-implemented CM in JAVA
- did not implement the entire CM
- just the core functionality of CM

# Syntactic Tools

- these tools are used to find
  syntactic information about programs
- what are the functions that *call* or are *called by* a
  particular function
- imagine a graph where each
  node represents a function

# Syntactic Tools

- there is an edge from `f` to `g` if `f` calls `g`
- the graph is called a *call graph*
- MIT students used a tool called `Imagix`
- used calls between files very effectively

## Semantic Tools

- syntactic tools do not infer any semantic information
- for example, syntactic tools do not perform any data-flow analysis
- what are the statements that effect a certain record directly or indirectly?

## Semantic Tools

- in the MIT project they used *Lackwit* and *CodeSurfer*
- will talk about both the tools in the next lecture
- check out the *CodeSurfer* home page at
  - `http://www.grammatech.com`
  - program slicing tool

# Other Tools Developed

- Concordance generator
  - web-interface to information gathered by program understanding
  - for each function stored
    - arguments and results
    - list of calling and called functions
    - optionally a two line specifications

# Other Tools Developed

- Message sequence chart generator
  - a post-processor to convert message traces into message sequence charts
  - visualize the flow of messages that happen in the CM
- a specialized script for handling messages

# Problems Found

- *Blocking Sends*
  - sending of messages uses blocking primitives
  - could cause CM to deadlock
- *Failures*
  - system was not fault-tolerant
  - CM goes down the entire CTAS crashes
- *Monitoring*
  - FAA would like to monitor the workings of CTAS
  - adding this to existing CM was difficult

# Lessons Learnt

- simple designs are possible
- standard software engineering techniques work
  - data abstraction
  - debate about where should software engineering education go
  - where should education cycles be spent?

# Lessons Learnt

- coding standards are vital
  - reverse engineering efforts helped by NASA's rigorous coding standards
  - good coding style helps the program analysis tools
  - consistent commenting criteria
- reverse engineering tools work
- high level models are vital
  - constructing object models