

# SOFTWARE FUNDAMENTALS: COLLECTED PAPERS - CH. 10, 26

Robert Dickerson

Department of Computer Science  
University of Virginia  
Charlottesville, Virginia 22904  
`rfd7a@cs.virginia.edu`

June 21, 2007

## 1 SOFTWARE ASPECTS OF STRATEGIC DEFENSE SYSTEMS

- Why the SDI software system will be untrustworthy
- Why conventional software development does not produce reliable programs
- The limits of software engineering methods
- Artificial Intelligence and the Strategic Defense Initiative
- Can automatic programming solve the SDI software problem?
- Can program verification make the SDI software reliable?

## 2 ON THE DESIGN AND DEVELOPMENT OF PROGRAM FAMILIES



# HISTORY

- Strategic Defense Initiative
  - started by Reagan to counter the threat of nuclear weapons
  - Anti ballistic missile (ABM)
- David Lorge Parnas
  - Panel of Computing in Support of Battlefield Management
  - resigned because software built would be untrustworthy
  - 20 yrs of research in software engineering
  - 8 yrs on real-time software for military aircraft



# REQUIREMENTS FOR A STRATEGIC DEFENSE SYSTEM

"I call upon the scientific community, who gave us nuclear weapons, to turn their great talents to the cause of mankind and world peace; to give us the means of rendering these nuclear weapons impotent and obsolete"

- Rapid and reliable warning of an attack
- Determination of the source of the attack
- Determination of the likely targets of the attack
- Determination of missile trajectories
- Coordination of the interception of the missiles
- Discrimination between decoys and warheads
- Detailed control of individual weapons
- Evaluation of the effectiveness of each attempt to destroy the target



## 8 ESSAYS

- There are fundamental differences between software engineering and other engineering disciplines
- Properties of an SDI software that make it unattainable
- Traditional techniques for building military software are inadequate
- Improvements in software engineering will not allow construction of reliable defense system
- Why research in AI will not help for SDI
- Why research in automatic programming will not help for SDI
- Why program verification (math proofs) will not give us reliable systems
- Why military spending of research in software and CS is inefficient and ineffective



# WHY SOFTWARE IS UNRELIABLE

- When most engineering products are completed, you expect it will work
- Not so for software products
  - Major "bugs"
  - Problems persist for several versions
  - "Improved" versions sometimes are worse
- People incorrectly point these problems on inept programmers



# SYSTEM TYPES

- *Discrete state* or digital systems
- *Continuous* or analog systems
- *Hybrid*



# MATHEMATICAL TOOLS

- Mathematics of continuous functions well understood
- No hidden surprises, small changes in inputs yield small changes in outputs
- With a small number of states, exhaustive testing is possible - compensates for lack of mathematical tools
- Math functions that describe the behavior of software are not continuous
- Traditional engineering cannot help in its verification
- Fundamental difference that will not disappear with improved technology \*



# HOW CAN WE UNDERSTAND SOFTWARE?

## EXAMPLE

### Solutions

- Build software as highly organized collections of small programs
- Use mathematical logic to replace continuous functions
- Division of software into modules help, but it is difficult to understand the complex conditions that arise from their interactions
- Large number of states and lack of regularity in the software yield very complex mathematical expressions
- These expressions are beyond the capacity of the human programmer and current computer systems



Why the SDI software system will be untrustworthy

## CHARACTERISTICS OF THE PROPOSED BATTLE-MANAGEMENT SOFTWARE SYSTEM

- System must be able to track ballistic characteristics with certainty (characteristics of decoys unknown)
- Impossible to test the system under realistic conditions prior to its actual use
- No possibility of debugging and modification to the program during period of service
- Resources available for computation cannot be predicted in advance
- Large variety of sensors and weapons the suite will be likely to grow, scalability a concern



Why the SDI software system will be untrustworthy

## MORE PROBLEMS

- No large-scale software system has been installed without extensive testing under realistic conditions
- We will be unable to make software modifications needed on the field (30-90 minute war for ICBM)



Why conventional software development does not produce reliable programs

## CONVENTIONAL METHOD

- Writing and understanding very large programs by "thinking like a computer" will be beyond our intellectual capabilities
- Programming is a trial and error craft
- Spend at least as much time testing code as writing it
- Software is not released for use when it is known as correct, but rather when the rate of discovering new errors slows down to an acceptable rate



# SOFTWARE ENGINEERING RESEARCH

- Structured programming and the use of formal program semantics
- Use of formally specified abstract interfaces to hide information
- Use of cooperating sequential processes to help deal with complexities arising from concurrency and multiprogramming



# SOFTWARE COST REDUCTION PROJECT

- Software requirements should be nailed down with complete black box requirements before software design is begun
- System should be divided into modules before coding
- Formal methods should be used for precise documentation
- Real-time systems should be built as set of cooperating sequential processes (with period and deadline)
- Programs should be written as structured programs



# WHAT MAKES SOFTWARE ENGINEERING HARD?

- Only when we have designed a similar system before is it easy to determine the requirements in advance
- Hard to make abstract interfaces until you understand all of the alternative designs
- New programming languages will not significantly help with these problems
- New programming environments will make small improvements
- Software engineering methods do not eliminate errors



# WHAT IS ARTIFICIAL INTELLIGENCE?

## EXAMPLE

- The use of computers to solve problems that previously could be solved only by applying human intelligence
  - The use of specific set of programming techniques known as heuristic or rule-based programming.
- 
- Rules from studying people turn out to be inconsistent, incomplete, and inaccurate.
  - Heuristic programs are developed by trial and error and a new rule is added when one finds a case that is not handled by old rules
  - Yields programs whose behavior is poorly understood and hard to predict
  - AI techniques requires experts who understand the problem, but we have no humans who can with high reliability and



Can automatic programming solve the SDI software problem?

# AUTOMATIC PROGRAMMING

- If we can write the specifications of the software, the computer will find the program
- Research in automatic programming is simply research in implementation of higher-level programming languages
- Automatic programming is feasible, but the efficiency of the resulting program is a concern
- Writing the specification is like writing the program itself
- No substantial change from present capability
- For SDI, we do not have the information to write specifications that we can trust



Can program verification make the SDI software reliable?

# PROGRAM VERIFICATION

- Programs are mathematical objects, meanings are mathematical objects, specifications are mathematical objects
- If we can prove programs correct, can't we prove SDI software correct?
- If it is proved correct, can we not rely on them in time of need?



Can program verification make the SDI software reliable?

## WHAT CAN WE PROVE?

- We can prove a small program meets a specification
- What is small?
  - In verification, a 500-line program is considered large
  - In SDI app, a 500-line program is considered small



Can program verification make the SDI software reliable?

## CAN WE HAVE FAITH IN PROOFS?

- Proofs increase our confidence in a program, but no basis for complete confidence
- Proofs reliable when they are small, well polished, and carefully read
- Proofs are practically restricted to sequential programs, not concurrent ones
- SDI software should be able to function if part of its equipment is destroyed by an enemy.



# PROGRAM FAMILIES

- Program families are sets of programs where there are so many common properties that it is advantageous to study the commonalities before analyzing the individual members.
- *Sequential development* is compared against *stepwise refinement* and *information hiding modules*
- Software will inevitably exist in many versions
  - variations in application demands
  - variations in hardware configurations
  - opportunity to improve a program
- Differences between versions are unavoidable and purposeful
- Cannot always design all algorithms before implementation of the system, need for experimental versions



# CLASSICAL METHOD OF PRODUCING PROGRAM FAMILIES (SEQUENTIAL COMPLETION)

- A particular member of a family is developed completely to the "working stage".
- The next member is developed by modification of the working programs.
- It is possible that descendants of a given program shares some of its ancestor's characteristics that aren't appropriate to the purpose of the descendants.
- Decisions remain in descendant program since their removal would be costly.
- The result is performance deficiencies, since programs were designed to function in a different environment or different load



# NEW TECHNIQUES

- Various versions need not be developed sequentially
- Development of one branch does not use information from another branch, two subfamilies could be developed in parallel
- By deciding as much as possible before the branching point we can increase the "similarity" of programs
- Certain decisions can be made without consideration of others (commutative)



# PROGRAMMING BY STEPWISE REFINEMENT

- Intermediate stages are represented by programs that are complete except for the implementation of certain operators and operand types
- Implementation of these operators are postponed to later stages

## EXAMPLE

```
begin variable table p; fill table p with first thousand numbers;
print table p; end
```



# TECHNIQUES OF MODULE SPECIFICATION

- Different than stepwise refinement in that intermediate representations are *not* incomplete programs.
- They are specifications of the externally visible behavior of program groups
- In the stepwise refinement, if you wanted to reconsider an earlier decision you would have to reconsider all decisions after that point
- So, in our example we can postpone the table implementation to a later stage (hide the decision)



# RELATION OF PROGRAM FAMILIES TO PROGRAM GENERATORS

- System generation programs
- Generator causes one member of the family to materialize to be loaded by target hardware
- Stepwise refinement and module specification can simplify the work on the generators
- System generators would be unnecessary if you were building a program which could simulate any member of the family
- Program might be relatively inefficient

