REUSE AT THE SOFTWARE PRODUCTIVITY CONSORTIUM

David M. Weiss
Software Productivity Consortium

The Software Productivity Consortium is sponsored by 14 aerospace companies as a developer of software engineering methods and tools. Software reuse and prototyping are currently the major emphasis areas. The Methodology and Measurement Project in the Software Technology Exploration Division has developed some concepts for reuse which they intend to develop into a synthesis process. They have identified two approaches to software reuse: opportunistic and systematic. The assumptions underlying the systematic approach, phrased as hypotheses, are the following: the redevelopment hypothesis, i.e., software developers solve the same problems repeatedly; the oracle hypothesis, i.e., developers are able to predict variations from one redevelopment to others; and the organizational hypothesis, i.e., software must be organized according to behavior and structure to take advantage of the predictions that the developers make. The conceptual basis for reuse includes: program families, information hiding, abstract interfaces, uses and information hiding hierarchies, and process structure. The primary reusable software characteristics are black-box descriptions, structural descriptions, and composition and decomposition based on program families. A good methodology has the following properties:

1. It answers the following key questions at any point in the development process:
   a. What should I do next?
   b. What output do I produce?
   c. What input and resources do I need to produce it?
   d. How do I know when I'm done?
2. It is based on a clear set of principles
3. It leads to quantifiable improvements in productivity and quality
4. It is useful to engineers
5. It promotes reuse
6. It supports a sound business approach

Automated support can be provided for systematic reuse, and the Consortium is developing a prototype reuse library and guidebook. The software synthesis process that the Consortium is aiming toward includes modeling, refinement, prototyping, reuse, assessment, and new construction. A number of key issues were also discussed.

TOPICS

- Concepts
  - Systematic vs Opportunistic Reuse
  - Assumptions Underlying Systematic Reuse
  - Underlying Principles
- Methodological Considerations
- Automated Support
  - Reusable Software Libraries
- Current Consortium Practice
- Direction
  - Synthesis

ORGANIZING SOFTWARE FOR REUSE

- Opportunistic Reuse – The Garage Sale Approach
  - Many individual parts
  - Search for part with desired behavior
    -- Attributes + Behavioral Description

- Systematic Reuse – Systems Approach
  - Collections of related parts
  - Search for system that meets requirements
    -- Attributes + Behavioral Description + Relationships + Classification

ASSUMPTIONS

- Redevelopment Hypothesis
  - Software developers solve same problems repeatedly
  - Solutions are captured as systems
  - Variations
    - -- Devices
    - -- Algorithms
    - -- Platforms
    - -- Functionality

- Oracle Hypothesis
  - Developer must be able to predict changes

- Organizational Hypothesis

  - Organize according to behavior and structure
  - Expose structures that make changeable decisions apparent
  - Identify common characteristics

DESIRABLE SOFTWARE CHARACTERISTICS

- Black-box description
  - Behavioral approach
- Structural Descriptions
- Composition and decomposition based on collections of parts

CONCEPTUAL BASIS

- Program Families
  - Characterize commonalities first
- Information Hiding
  - Encapsulate changeable decisions
- Abstract Interfaces
  - Behavioral descriptions of modules
- Uses Hierarchy
  - Protect subsettability
  - Explicit decisions about dependencies
- Information Hiding Hierarchy
  - Roadmap for change
- Process Structure
  - Performance assessment
  - Reconfigurability

REUSABLE SOFTWARE CHARACTERISTICS

- Black-Box Descriptions
  - Behavioral approach, e.g., based on A-7 module descriptions
- Structural Descriptions
  - Hierarchical views, based on information hiding and uses hierarchies
- Composition and Decomposition Based On Program Families
  - Families described structurally
  - Components of families described behaviorally
  - Many shared subfamilies

METHODOLOGICAL CONSIDERATIONS

DEFINITIONS

- PROCESS
  - Set of activities used to produce and maintain software
- METHODOLOGY: Answers to the Questions:
  - What do I do next?
  - What input do I need to do it?
  - What output do I produce?
  - What resources do I need to produce it?

ATTRIBUTES OF A GOOD METHODOLOGY

- Answers the key questions
  - What do I do next?
  - What input do I need to do it?
  - What output do I produce?
  - What resources do I need to produce it?
- Based on a clear set of principles
  - Information hiding, hierarchical structuring, etc.
- Leads to improvements in productivity and quality
  - Measurable
- Useful to engineers
- Promotes reuse
- Supports sound business approach
- Can be adopted incrementally

OBJECTS SUPPORTING SYSTEMATIC REUSE

- Requirements Specification
- Module Guide
- Abstract Interface Specification
- Allowed-to-Use Hierarchy
- Module Internal Documentation
- Uses Hierarchy
- Process Structure
- Potential Family Members

} Design

- Code
- Tests

SOFTWARE EVOLUTION PROCESS

- Developers maintain collections of program families
- Requirements are identified using families that support:
  - simulation, prototyping, modeling, other forms of analysis,
  - production of specifications.
- Given the requirements for a new system:
  - the collections are searched for a family with a member that meets the requirements,
  - modules of the family are adapted and assembled to produce the new member,
  - if no such family exists, a new family is created (rare).

AUTOMATED SUPPORT FOR SYSTEMATIC REUSE

- Reuse Library
  - Repository for collections of families
- Adaptation Analysis
  - Tracing the effects of change
- Construction of specifications
  - Editor, browser
  - Design representation
- Adaptation Mechanism
  - Parameterized module/subset generation
    - -- Generic, macros
- Modeling
  - Performance analysis
- Composition
  - System generation

REUSE LIBRARIES

- Storage of Life Cycle Objects (LCOs) and their Descriptions
  - Requirements
  - Module Guide
  - Module Interface Specification
  - Code
  - :
  - :
  - :
- Search Mechanisms
  - By Attribute
    - -- Language, version, author, producing tool, etc.
  - By LCO type
  - By Relation
    - -- Hierarchy traversal (uses, information hiding, composition, etc.)
  - By Classification
- Object descriptions
- Population

CURRENT CONSORTIUM PRACTICE

- Guidebook
  - Management and Technical Volumes
- Reuse Library Prototype

Software Technology Exploration Guidebook
Volume 1: Management Guidebook

Table of Contents

August 26, 1988

Software Technology Exploration Guidebook
Volume 2: Technical Guidebook

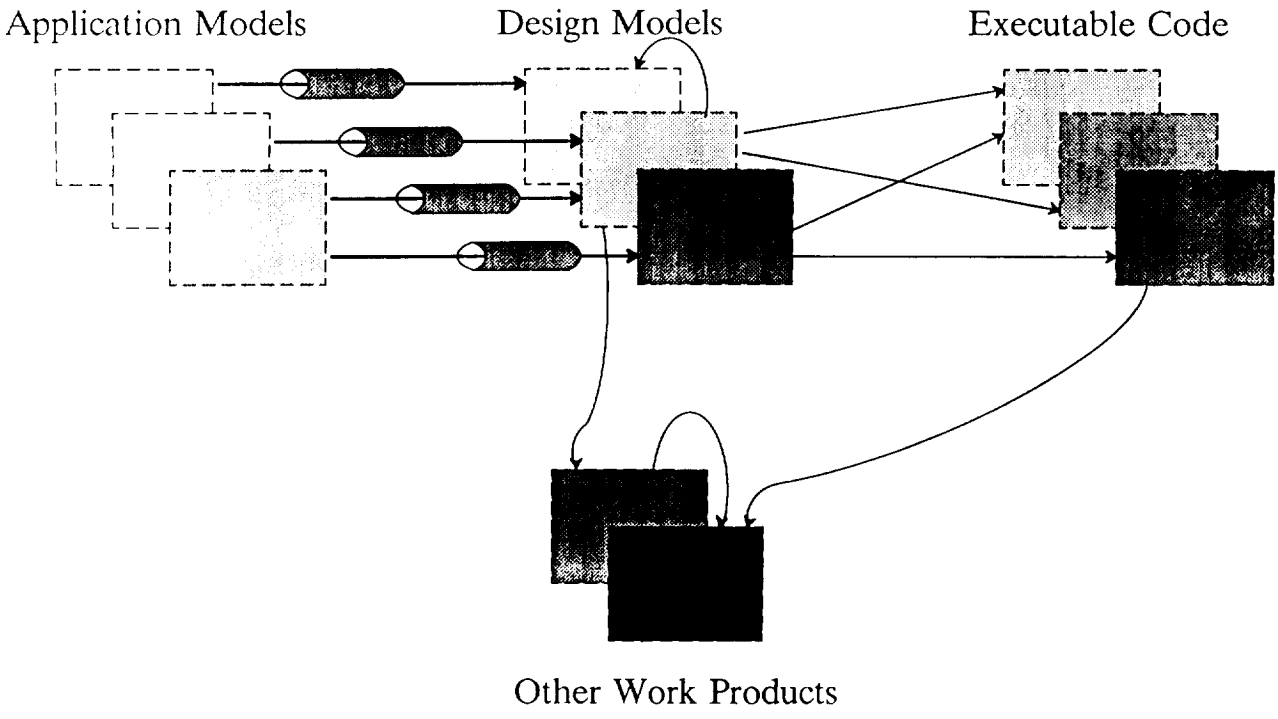
Table of Contents

August 31, 1988

DIRECTION

SYNTHESIS: A PROCESS THAT RELIES ON THE PRODUCTION OF SOFTWARE FROM MODELS SPECIFICALLY DESIGNED FOR REUSE

RELATIONSHIPS AMONG MODELS



ASPECTS OF SYNTHESIS

- MODELLING
  - APPLICATION
  - DESIGN
  - IMPLEMENTATION
- REFINEMENT
  - SUCCESSIVE APPROXIMATION OF PROBLEM
  - SUCCESSIVE APPROXIMATION OF SOLUTION
- PROTOTYPING
  - REFINEMENT THROUGH ISSUE RESOLUTION
- REUSE
  - STANDARDIZED ENGINEERING SOLUTIONS
  - SOLUTIONS REPRESENTED IN TERMS OF REUSABLE PARTS
  - COMPOSITION OF NEW SYSTEMS FROM EXISTING, ADAPTED, AND NEW PARTS
- ASSESSMENT
  - QUANTIFICATION OF APPROXIMATION
- NEW CONSTRUCTION

STEPS IN THE SYNTHESIS PROCESS

- SPECIFY REQUIREMENTS
  - DIRECTLY, IN TERMS OF PRE-DEFINED DOMAIN VOCABULARY
  - ANALOGOUSLY, IN TERMS OF DIFFERENCES BETWEEN NEW NEEDS AND EXISTING SYSTEMS
- MAKE APPLICATION MODEL
  - MODEL CONSTRUCTION
  - MODEL ASSESSMENT
- MAKE DESIGN MODEL
  - SELECTION OF CANONICAL DESIGN
  - ADAPTATION OF CANONICAL DESIGN
  - INVENTION OF NEW DESIGN
  - ASSESSMENT OF DESIGN
- IMPLEMENT
  - COMPLETION OF NEW AND ADAPTED PARTS
  - COMPOSITION OF PARTS INTO PROTOTYPES/PRODUCTS
  - VERIFICATION

SUPPORTING ELEMENTS

- REPOSITORIES
  - REUSE LIBRARY
  - PROJECT LIBRARY
- REPRESENTATION TECHNOLOGY
  - USER INTERFACE
  - SPECIFICATIONS FOR MODELS, DESIGNS, CODE PRODUCTION
- METHODOLOGY
  - PROCESS MODEL
  - NATURE OF PARTS AND RELATIONS
  - MANAGEMENT OF PROCESS
- ARCHITECTURE OF TOOLSETS
  - TOOLSET INTEGRATION

KEY ISSUES

- HOW TO DO DOMAIN ANALYSIS
  - SYSTEMATIC APPROACH
  - APPLICATION MODELLING
  - RE-ENGINEERING
- NOTATIONS AND MECHANISMS FOR MAPPING FROM APPLICATION MODEL TO SOFTWARE DESIGN
  - WHAT NOTATIONS?
  - HOW MANY INTERMEDIATE LEVELS?
- HOW TO REPRESENT DESIGNS: PARTS AND THEIR RELATIONS
  - WHAT NOTATIONS?
  - STORAGE, RETRIEVAL, AND SEARCH
  - RE-ENGINEERING
- HOW TO ADAPT, COMPOSE, AND VERIFY PARTS
  - DESIGN PARTS
  - CODE PARTS

KEY ISSUES (CONC)

- HOW TO ASSESS DESIGNS AND CODE
  - PERFORMANCE
  - FUNCTION
  - DEPENDABILITY
- HOW TO PRODUCE CODE
  - PROTOTYPE
  - PRODUCTION

- HOW SHOULD THE ENGINEERS INTERACT IN THE PROCESS?
  - INTERFACE
  - PROCESSING STEPS
  - UNDERSTANDING OF CONTEXT
- HOW SHOULD THE PROCESS BE MANAGED?
- HOW SHOULD THE EFFECTIVENESS BE MEASURED?
  - CURRENT STATE
  - GUIDE TO IMPROVEMENT

PROJECTS

- METHODOLOGY, MEASUREMENT, AND MANAGEMENT
- DESIGN REPRESENTATION, MAPPING FROM APPLICATION MODELS, AND COMPOSITION
- DOMAIN ANALYSIS
- REPOSITORIES
- ASSESSMENT
- VERIFICATION
- ADAPTATION
- INTERACTION WITH THE ENGINEERS

PROGRAM FAMILIES

HARDWARE ANALOGIES -- THE IBM 360, DEC PDP-11

- Families of computers
  - Same instruction set architecture (Behavioral description)
  - Different implementations
  - Same operating system (Different versions)
- Program family: A set of programs is a program family if the programs have so much in common that it pays to study their common characteristics before investigating the special properties of individual programs.

EXAMPLE FAMILY CLASSIFICATION

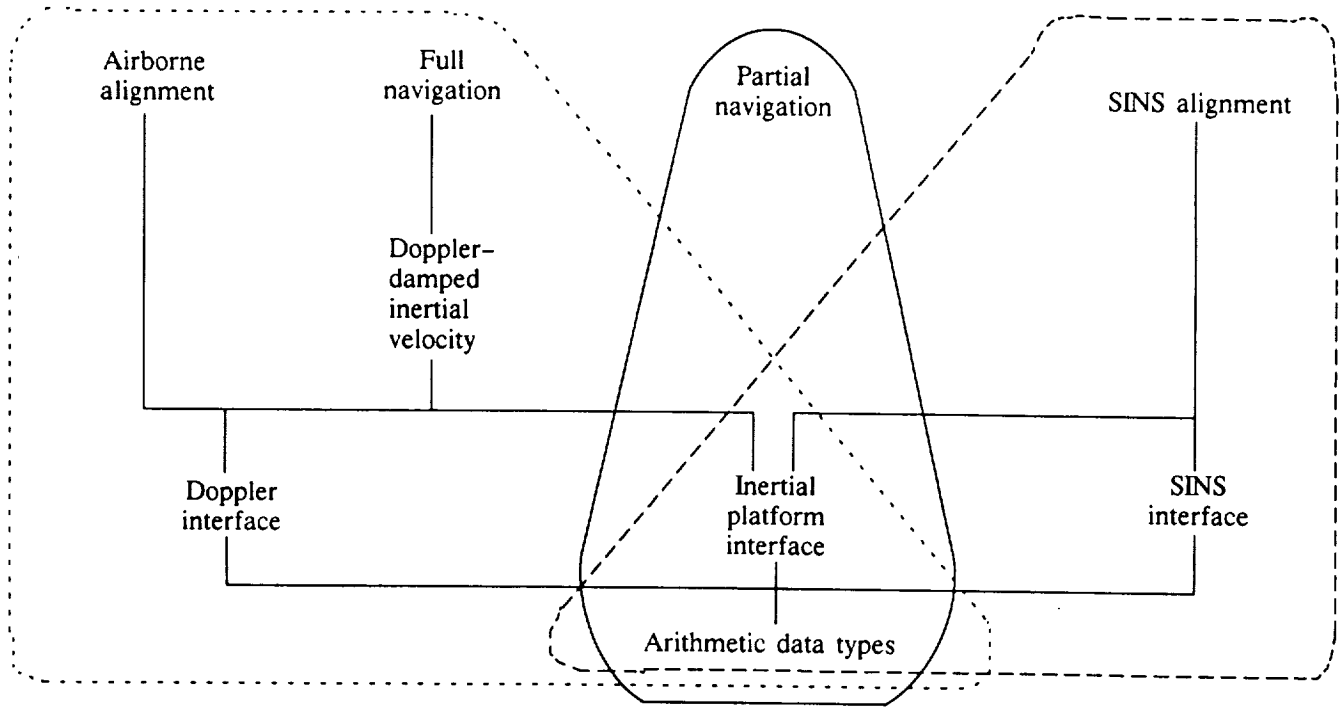
Tool Families

Process Support	Technology
Describing	Data Storage & Retrieval
Composing & Decomposing	User Interface
Assessing	Data Transformation
Retaining	Configuration

Application Families

Avionics	Communications
A-6	Control Systems
A-7	Speech Processing
727	
737	

A-7 NAVIGATION FAMILY



INFORMATION HIDING

- A decision that is likely to change should be encapsulated in a module
  - Changeable decision is the secret
    - -- data structures
    - -- algorithms
    - -- device characteristics
  - Basis for Ada packages
- Module: An information hiding module is a work assignment
  An information hiding module is a black-box
  An information hiding module is a set of programs and shared data
  An information hiding module is a finite state machine
- Interface: An interface between two modules is the set of assumptions that the programmer of one module may make about the other module
- Abstract Interface: An interface that represents many possible actual interfaces

HIERARCHIES AND STRUCTURE

- Module Hierarchy

  Parts: Modules
  Relation: Subsecret — B is a submodule of A if B's secret is a subsecret of A's secret

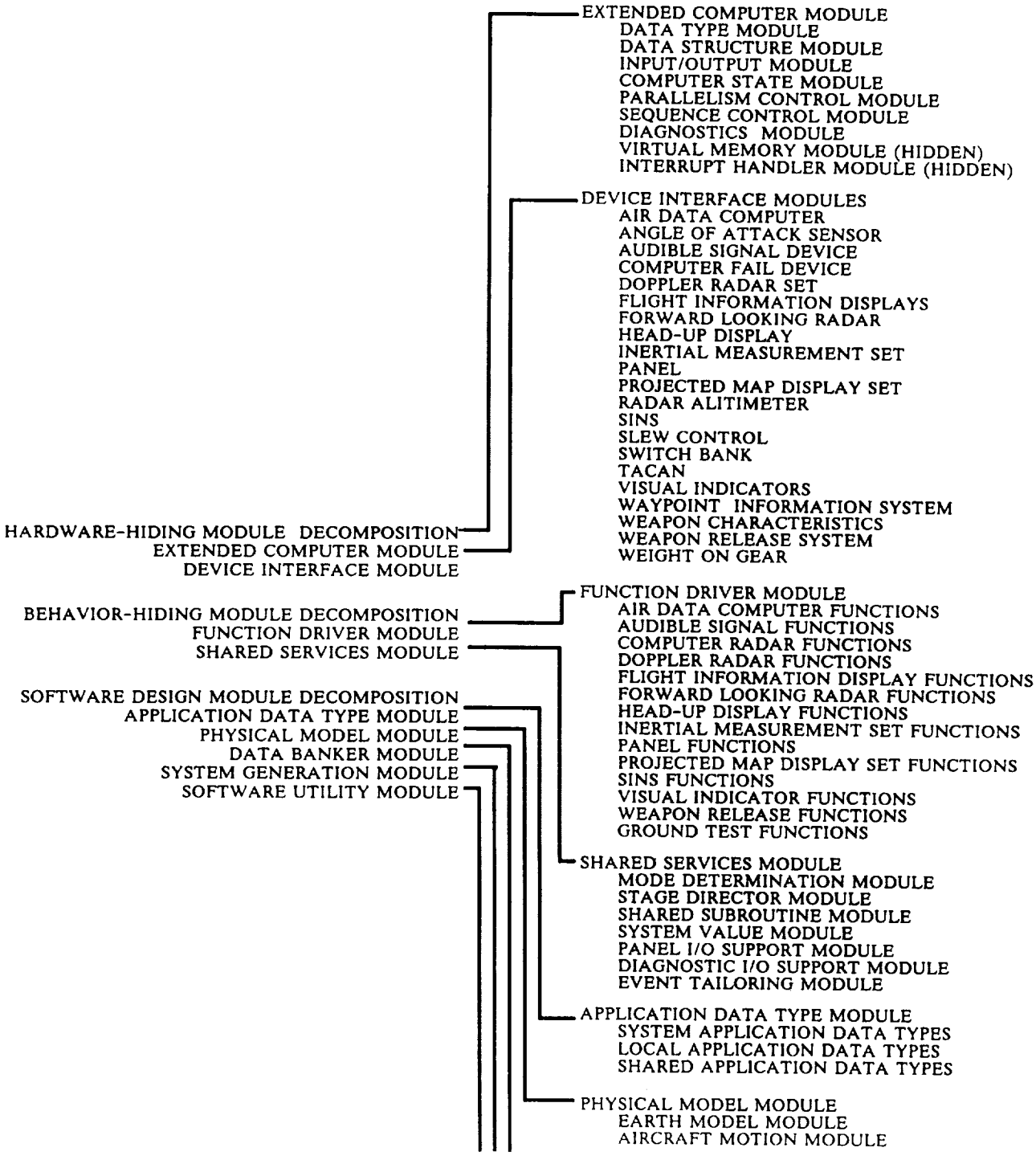
- Uses Hierarchy

  Parts: Programs
  Relation: Uses — A uses B if A requires the presence of B

- Process Structure

  Parts: Processes
  Relation: Awaits — A awaits B if A cannot progress until B progresses

A-7 MODULE STRUCTURE



CHARACTERIZING FAMILIES

- By structure
  - module hierarchy, uses hierarchy, process structure
- By function
  - navigation, device control
- By application
  - avionics, satellite communications
- By technology
  - relational database, MonteCarlo simulation
- By ???

- Families composed of modules
- Modules characterized by externally-visible behavior
  - Black-boxes
  - Abstract interfaces specify behavior

SUMMARY

- Systematic Reuse Based on Families
- Concepts
  - Program Families (1976)
  - Information hiding (1970)
  - Abstraction
  - Behavioral specification (1972)
  - Hierarchies
- Examples
  - A-7 (1980)
- Relations to other technology
  - Ada
  - Object oriented programming
  - Object oriented design