JOHNSON GRANT

IN-61-CR

308262

(NASA-CR-187266) SOFTWARE ENGINEERING AND THE ROLE OF Ada: EXECUTIVE SEMINAR (Houston Univ.) 140 p CSCL 09B

N91-13087

Unclas

G3/61 0308262

Software Engineering and the Role of Ada

Executive Seminar

Glenn Freedman

University of Houston - Clear Lake

May 31, 1987

Cooperative Agreement NCC 9-16

Research Activity No. *ET.1*

Research Institute for Computing and Information Systems

University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the "gateway" concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.

Software Engineering and the Role of Ada

Executive Seminar

Preface

This research was conducted under the auspices of the Research Institute for Computing and Information Systems by Glenn Freedman, founding Director of the Software Engineering Education Center (SEPEC) of the University of Houston - Clear Lake.

Funding has been provided by the Spacecraft Software Division, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston - Clear Lake. The NASA Technical Monitor for this activity was Steve Gorman, Deputy Chief of Space Station Office, Mission Support, NASA/JSC.

SOFTWARE ENGINEERING AND THE ROLE OF ADA*

EXECUTIVE SEMINAR

UNIVERSITY OF HOUSTON- CLEAR LAKE

SOFTWARE ENGINEERING AND ADA TRAINING PROJECT

*Ada is a registered trademark of the U.S. Government, AJPO

PROGRAM AGENDA

I. THE SOFTWARE CRISIS: PROBLEMS AND SOLUTIONS

II. MANDATE OF THE SPACE STATION PROGRAM

III. THE SOFTWARE LIFE CYCLE

IV. SOFTWARE ENGINEERING

V. ADA UNDER A SOFTWARE ENGINEERING UMBRELLA

PROGRAM GOALS

* Review the software life cycle

* Apply the concepts of current software engineering to space station issues

* Examine the role of Ada+ language in the software development environment

+ Ada is a trademark of the US Government, Ada Joint Program Office

THE SOFTWARE CRISIS

PROBLEMS AND SOLUTIONS

THE SOFTWARE CRISIS

KEY ELEMENTS

* Over budget and late
* Actual life cycle cost
* Modification is difficult, time consuming and costly
* The software invasion

COST OF SOFTWARE =

Original development cost

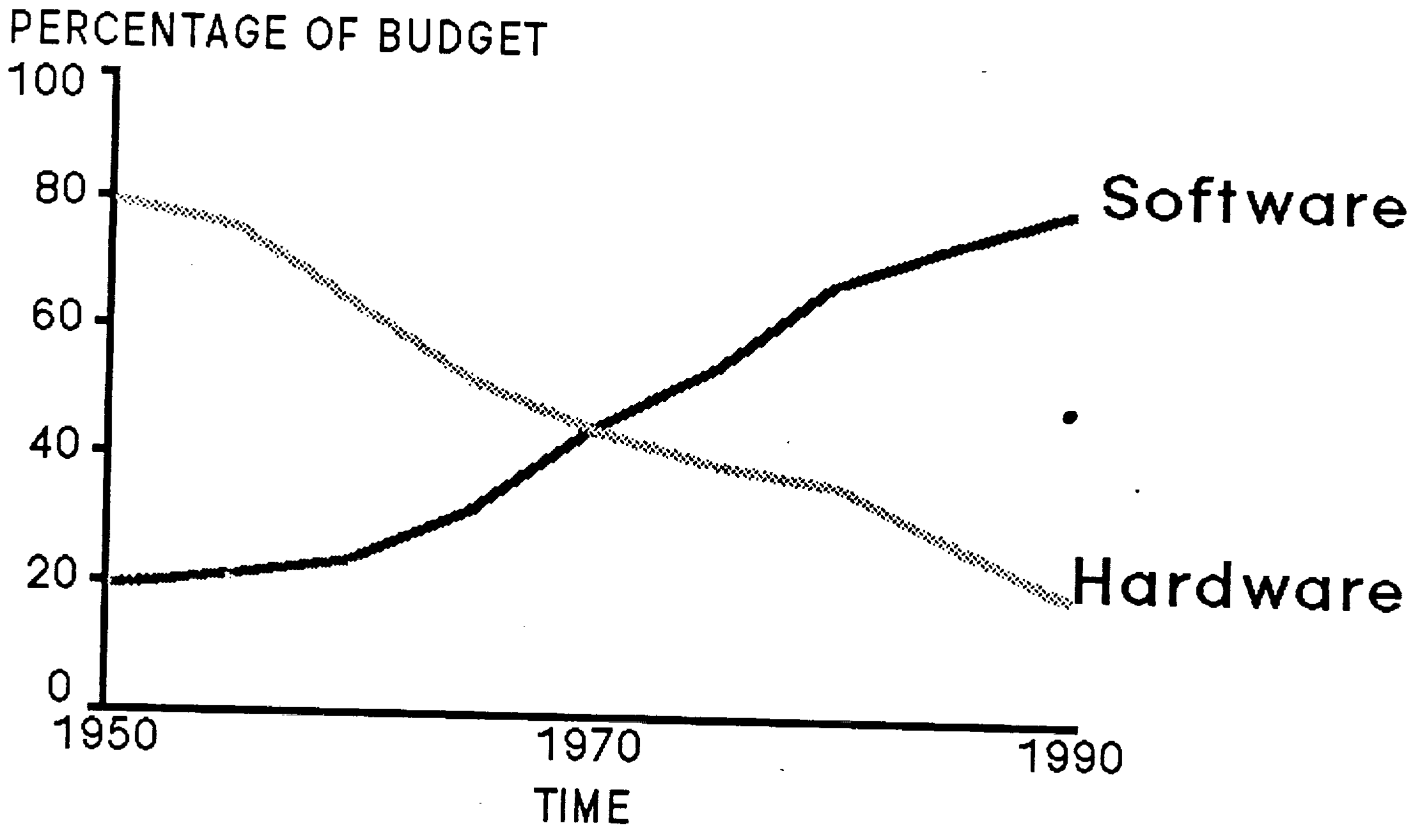
+

Maintenance/Modification costs

+

Unanticipated costs

The High Cost of Software



HIGH PROJECT COSTS

REASONS

* Poor programming techniques
* Poor design and specification techniques
* Improper choice of language for job

HIGH PROJECT COSTS

PARTIAL SOLUTIONS

* Structured programming (mid '60s)
* Software Engineering
  - Measurement tools:
    - Cohesion
    - Coupling
    - Fan-in/Fan-out
    - Factoring
  - Design Techniques
    - Top Down Design
    - Data Flow Design
    - "Structured" Design
    - Object Oriented Design

HIGH PROJECT COSTS

PARTIAL SOLUTIONS

* Improvements in language design and development of specialized languages
  - Pascal
  - "C"
  - Prolog

MANDATE OF THE SPACE STATION PROGRAM

PROFILE OF SPACE STATION PROGRAM

* Large
* Complex
* Distributed Networks

PROFILE OF SPACE STATION PROGRAM

* Embedded Components
  - Parallel Processing
  - Real Time Control
  - High Reliability
  - Safety
  - Non-Stop Operation
* Long-Term Life Expectancy
* Over 100 million lines of code

SOFTWARE CHALLENGE

* Many needs initially undetermined and unknown

* Many requirements initially undefined

* Personnel continuity an unrealistic goal

* Vendor continuity an unrealistic goal

SOFTWARE CHALLENGE

* Many needs are never fully determined - always changing

* Integration of new functions in an incrementally evolving system

WHAT ARE THE SOFTWARE REQUIREMENTS?

* Modifiability
* Efficiency
* Reliability/Safety
* Understandability
* Correctness
* Portability/Interoperability /Extensibility

SOFTWARE MUST BE MODIFIABLE AND EFFICIENT

DEFINITIONS

MODIFIABILITY is the ability to control change within software, thus achieving new results without undesirable or disastrous side effects.

EFFICIENCY is the extent to which software performs its intended functions with a minimum of consumption of computing resources.

SOFTWARE
MUST
BE
RELIABLE
AND
SAFE

DEFINITIONS

RELIABILITY is the ability of a program to perform a required function under stated conditions for a stated period of time.

SAFETY is the ability of software to protect life and property in the presence of "N" faults.

SOFTWARE
MUST
BE
UNDERSTANDABLE
AND
CORRECT

DEFINITIONS

UNDERSTANDABILITY is the extent to which the software's algorithms and data structures are easily perceived and easily interpreted.

CORRECTNESS is the extent to which software is free from design defects and from coding defects - that is fault free, the extent to which software meets its specified requirements and the extent to which software meets user expectations.

SOFTWARE
MUST
BE
PORTABLE,
INTEROPERABLE AND
EXTENSIBLE

DEFINITIONS

PORTABILITY is the ease with which software can be transferred from one computer system or environment to another.

INTEROPERABILITY is the ability to "use" the entities that are "ported" among systems and the properties of the entities, the relationships to other entities, and the properties of these relationships.

DEFINITIONS

EXTENSIBILITY is the result of models and rules which allow controlled changes with predictable effects to be made to both interfaces and the models of services and resources on any side of the interfaces.

SOFTWARE
LIFE
CYCLE

WHAT IS THE SOFTWARE LIFE CYCLE?

DEFINITION

SOFTWARE LIFE CYCLE

A software engineer's model of the activities and phases involved in the processes of producing and sustaining a system's software products from conception through retirement.

NASA SOFTWARE ACQUISITION LIFE CYCLE MODEL

* Software Concept & Project Definition
* Software Initiation
* Software Requirements Definition
* Software Architecture Design
* Software Detail Design

NASA SOFTWARE ACQUISITION LIFE CYCLE MODEL

* Software Implementation
* Software Systems Integration and Testing
* Software Acceptance Testing and Delivery
* Operation and Maintenance Transition

SUSTAINING ENGINEERING ACTIVITIES

* System Requirements Analysis
* Software Requirements Analysis
* Preliminary Design
* Detailed Design
* Coding and Unit Test
* Computer Software Component Integration

SUPPORTING ACTIVITIES

* Documentation
* Configuration Management and Integration Control
* Quality Management
* Review
* Verification & Validation
* Communication Through the Project Object Base

SUPPORTING ACTIVITIES

* Automated Support
  - Technical Tools
  - Management Tools

IMPACT OF CHANGE ACROSS LIFE CYCLE

PROBLEMS

* Time
* Money

SOFTWARE ENGINEERING

DEFINITION

"SOFTWARE ENGINEERING IS THE ESTABLISHMENT, AND APPLICATION OF SOUND ENGINEERING CONCEPTS, PRINCIPLES, MODELS, METHODS, TOOLS AND ENVIRONMENTS TO SUPPORT COMPUTING WHICH IS:

CORRECT
MODIFIABLE
RELIABLE
EFFICIENT
UNDERSTANDABLE

THROUGH THE LIFE CYCLE OF THE APPLICATION."

(C. MCKAY, 1985)

IMPACT OF CHANGE ACROSS LIFE CYCLE

SOLUTIONS

* Early Error Detection
* Reusable Components
* High Quality Documentation
* Automated Tools and Methods

WHY SOFTWARE ENGINEERING

DISCIPLINED APPROACH TO SOFTWARE DEVELOPMENT AND MAINTENANCE USING:

* Proven Management Techniques
* Proven Technical Methods

COMPUTER SCIENCE

⬇

⬇

⬇

SOFTWARE ENGINEERING

* Modifiability
* Efficiency

GOALS OF SOFTWARE ENGINEERING

* Reliability
* Correctness
* Understandability

DEFINITION

MODIFIABILITY

Modifiability is the ability to control change within software, thus achieving new results without undesirable or disastrous side effects.

MODIFIABILITY

KEY ELEMENTS

* Controlled change
* Change without surprises
* Change without unpredictable side effects

MODIFIABILITY

IMPLICATIONS

* Encapsulation of Code and Design
* Generic, Reusable Units
* Time Requirements

DEFINITION

EFFICIENCY

Efficiency is the extent to which software performs its intended function with a minimum consumption of computing resources.

EFFICIENCY

KEY ELEMENTS

* Producing the desired result with a minimum of effort or waste
* Making optimal use of available resources: space, time, people, etc.

EFFICIENCY

IMPLICATIONS

* Requires some compromises
  - Time/Space
  - Reliability/Time

DEFINITION

RELIABILITY

Reliability is the ability of a program to perform a required function under stated conditions for a stated period of time.

RELIABILITY

KEY ELEMENTS

* Runs Well

* Fails Gracefully

RELIABILITY

IMPLICATIONS

* Need for enforced standards

* Need for normal and exception modes of operation

DEFINITION

UNDERSTANDABILITY

Understandability is the extent to which the software's algorithms and data structures are easily perceived and easily interpreted.

UNDERSTANDABILITY

KEY ELEMENTS

* Systems can be understood in appropriate detail throughout the life cycle

* Critical goal in management of complex systems

UNDERSTANDABILITY

KEY ELEMENTS

* Development engineers will not be the sustaining engineers
* In a large, complex, non-stop, distributed system which evolves incrementally over more than 20 years, a principal challenge will be integration control.

UNDERSTANDABILITY

IMPLICATIONS

* Design Decisions
* Documentation Standards
* Language Selection

DEFINITION

CORRECTNESS

Correctness is the extent to which:

* software is free from design and coding defects - that is fault free
* software meets its specified requirements
* software meets user expectations

CORRECTNESS

KEY ELEMENTS

* The software successfully meets the requirements as written
  - Functional Requirements
  - Non-functional Requirements

CORRECTNESS

IMPLICATIONS

* Normal operations are considered
* Exception conditions are considered
* Software can be verified and validated
  - Verification - Are we building it right?
  - Validation - Did we build the right thing?

SOFTWARE ENGINEERING PRINCIPLES

SOFTWARE ENGINEERING PRINCIPLES

* ABSTRACTION
* INFORMATION HIDING
* MODULARITY
* LOCALIZATION
* CONFIRMABILITY
* COMPLETENESS
* UNIFORMITY

DEFINITION

ABSTRACTION

Abstraction is an intellectual tool that allows one to deal with conceptual aspects of a software system apart from the implementation details allowing an overview of an entire system or its components.

ABSTRACTION

KEY ELEMENTS

* Limit amount of detail
* High Levels -- minimum detail
* Top-down Design
* Essential information only
* Focus on WHAT not HOW - separate the spec from implementation

DEFINITION

INFORMATION HIDING

Information hiding is the process which removes all unnecessary details from a user's access thereby protecting the software system from unexpected or unwanted changes.

INFORMATION HIDING

KEY ELEMENTS

* "What" is visible (in Spec)
* "How" is hidden (in Implementation)
* Makes certain details inaccessible
* Protects implementation from accidental corruption

DEFINITION

MODULARITY

Modularity is the purposeful structuring of elements (or software modules) that are integrated to satisfy system requirements (loosely coupled).

MODULARITY

KEY ELEMENTS

* Logical division into stand alone units
* Units have specific function and clearly defined interfaces
* Discrete components
* Change to one component has minimal impact on other components

DEFINITION

LOCALIZATION

Localization is the process of creating strongly cohesive programming units, that is, locating elements which exhibit a high degree of functional relatedness within one unit.

LOCALIZATION

(Separation of Concerns)

KEY ELEMENTS

* Logically related pieces
* Cohesive - internally tight
* Loose connection between modules
* Independent - loosely coupled
* Allows firewalling of the effects of errors, i.e., prevents errors within one module from affecting other modules.

DEFINITION

CONFIRMABILITY

Confirmability is the evaluation of the software system and its components from a requirements perspective or a design perspective.

CONFIRMABILITY

KEY ELEMENTS

* Can be decomposed and tested
* Documentation through all of the life cycle phases, including design decisions and rationale

DEFINITION

COMPLETENESS

Completeness is the process of ensuring that all design elements are present in the system.

COMPLETENESS

KEY ELEMENTS

All important elements specified in the requirements and the design are present

DEFINITION

UNIFORMITY

Uniformity is the degree to which consistent notation is used.

UNIFORMITY

KEY ELEMENTS

* Consistency across life cycle
* Standardization in:
  - Language
  - Documentation
  - Coding Style
  - Conventions

SOFTWARE ENGINEERING TOOLS AND METHODS

DEFINITION

LIFE CYCLE

The issues of creating, building and sustaining any system from conception to retirement.

DEFINITIONS

SOFTWARE ENGINEERING TOOLS AND METHODS

TOOLS - APPLY AUTOMATION TO SOFTWARE DEVELOPMENT WITHIN THE CONTEXT OF THE METHOD.

METHODS - PROVIDE A SYSTEMATIC APPROACH INDICATING HOW TO DEVELOP INTERMEDIATE SOFTWARE PRODUCTS WITHIN THE CONTEXT OF THE LIFE CYCLE MODEL.

SOFTWARE ENGINEERING TOOLS

SOFTWARE ENGINEERING TOOLS

Program Design Language (PDL)

* Can be compiled
  - Early error checking
  - Early interface checking
* Allows for decomposition of problem
* Design is visible early
  - Limits risks
* Flows naturally into code
* Possible drawback:
  - Tendency to focus on detail not design

SOFTWARE ENGINEERING TOOLS

EXAMPLES OF OTHER TOOLS

- Languages
- Editors
- File Managers
- Debugging Tools
- Complexity Analyzers
- Report Generators

SOFTWARE ENGINEERING METHODS

SOFTWARE ENGINEERING METHODS

STRUCTURED ANALYSIS AND DESIGN TECHNIQUE (SADT)

DEVELOPED BY DOUG ROSS OF SOFTECH IN THE EARLY '70S. THIS IS A MANUAL SYSTEM WHICH COULD BE AUTOMATED.

FEATURES:
* FORMAL BLOCK DESIGN
* SIMPLE
* CLEAR
* SUPPORTS MODULARITY

DRAWBACK:
* WITHOUT AUTOMATION IT IS TEDIOUS TO KEEP CURRENT

SOFTWARE ENGINEERING METHODS

STRUCTURED DESIGN

* FOCUS IS ON ALGORITHMS AND OPERATIONS

* WIDELY USED IN FORTRAN APPLICATIONS

SOFTWARE ENGINEERING METHODS

JACKSON'S DATA FLOW DESIGN

* Focus is limited to the data structure
* Data driven design
* Widely used in COBOL applications

SOFTWARE ENGINEERING METHODS

OBJECT ORIENTED DESIGN (OOD)

* Method:
  - Select/Develop informal strategy
  - Identify objects and operations on those objects
  - Tool: Ada

SOFTWARE ENGINEERING METHODS

OBJECT ORIENTED DESIGN (OOD)

* Approach
  - Considers data structures and algorithms as a unit - object
  - Separate WHAT from HOW

ADA UNDER A SOFTWARE ENGINEERING UMBRELLA

QUESTION:

WHY WAS ADA DEVELOPED?

SOFTWARE WAS:

* COSTLY
* UNRESPONSIVE
* UNRELIABLE
* LATE
* UNMODIFIABLE
* NON-PORTABLE
* INEFFICIENT
* POTENTIALLY UNSAFE

ADA UNDER A SOFTWARE ENGINEERING UMBRELLA

RATIONALE FOR DEVELOPMENT

* Costs up
* Quality down
* Changing needs

SOFTWARE ENGINEERING AND THE ROLE OF ADA

* Overall life cycle costs must be reduced
* New approaches are needed.to meet the software challenge of the future and growing life cycle issues
* It is imperative to identify sound software engineering strategies
* Software engineering techniques must be applied across the life cycle

THE HISTORY OF ADA

REQUIREMENT DEFINITION PHASE

HOWLG: Higher Order Language Working Group (DOD)

STRAWMAN: First draft of requirements for DOD's programming language

WOODENMAN: Comment on Strawman

TINMAN: Comment on Woodenman

IRONMAN: Comment on Tinman

STEELMAN: Comment on Ironman

THE HISTORY OF ADA

REQUIREMENT DEFINITION PHASE

RFP's solicited to design language.

4 Proposals selected to proceed.

THE HISTORY OF ADA

REQUIREMENT DEFINITION PHASE

STEELMAN: Final language requirements document.

DOD 5000.29: Use only DOD approved language in defense systems.

DOD 5000.31: Listed approved higher order languages.

THE HISTORY OF ADA

DESIGN TEAM SELECTION

7/78 -- Blue Team: SofTech
Yellow Team: SRI International
Red Team: Intermetrics
Green Team: Honeywell Bull

11/78 -- Red Team: Intermetrics
Green Team: Honeywell Bull

5/79 -- Green Team: Honeywell Bull
Team Leaders: J. Ichbiah
J. Barnes
R. Firth

THE HISTORY OF ADA

NAMING THE LANGUAGE (MAY 1979)

* Ada Lovelace (1815-1851)
  - Worked with Charles Babbage on his difference and analytic engines
  - Considered the world's first programmer
  - Augusta Ada Byron,
    Countess of Lovelace,
    Daughter of poet Lord Byron

THE HISTORY OF ADA

ENVIRONMENTAL REQUIREMENTS

SANDMAN: Initial analysis of environment requirement.

PEBBLEMAN: Revised environment requirement.

STONEMAN: Final environment requirement.

THE HISTORY OF ADA

MILESTONES

* ACV - Ada Compiler Validation
* AJPO - Ada Joint Program Office
* LRM - Language Reference Manual
  January 1983
* ANSI MILSTD 1815A (February 1983)

ADA UNDER A SOFTWARE ENGINEERING UMBRELLA

DE LAUER PRONOUNCEMENT (1983)

"...THE ADA PROGRAMMING LANGUAGE SHALL BECOME THE SINGLE, COMMON COMPUTER PROGRAMMING LANGUAGE FOR DEFENSE MISSION-CRITICAL APPLICATIONS. EFFECTIVE 1 JANUARY 1984 FOR PROGRAMS ENTERING ADVANCED DEVELOPMENT AND 1 JULY 1984 FOR PROGRAMS ENTERING FULL-SCALE DEVELOPMENT, ADA SHALL BE THE PROGRAMMING LANGUAGE... ."

ADA FEATURES

* Strong Specification
* Strong Typing
* Tasks
* Generics
* Exception Handlers
* Packages

DEFINITION

SPECIFICATION

* "A specification is a document that prescribes in a complete, precise and verifiable manner the requirements, design, behavior or other characteristics of a system or system components." (IEEE, 1983)

STRONG SPECIFICATION

KEY ELEMENTS

* All program units have a declared interface or specification.

* Ada enforces compliance with this interface.

DEFINITION

TYPING

A type characterizes both a set of values and a set of operations on those values.

STRONG TYPING

KEY ELEMENTS

* Ada is a strongly typed language
* All objects (variables and constants) in Ada must have a type
* A type defines:
  - A set of values
  - A set of operations allowed

DEFINITION

TASK

A task is a program unit that may execute in parallel with other program units.

TASKS

KEY ELEMENTS

* An Ada task operates in parallel with other Ada program units
* Tasking provides parallel processing
  - Single Processor Computers
  - Multi Processor Computers
  - Distributed Networks of Computers

TASKS

KEY ELEMENTS

* An Ada task operates in parallel with other Ada program units
* Tasking provides parallel processing
  - Single Processor Computers
  - Multi Processor Computers

DEFINITION

GENERICS

Generics are parameterized templates of a program unit that allow reuse of code and that allow libraries of programs to be built.

GENERICS

KEY ELEMENTS

* Generic unit is a template or mold for other program units - a set of subprograms or a set of packages

* Generics are not executable

GENERICS

KEY ELEMENTS

* Formal parameters (those in the template) are replaced with actual parameters when it is used

DEFINITION

EXCEPTION HANDLERS

An exception handler is code that tells the program what to do if an exceptional situation or error occurs.

GENERICS

KEY ELEMENTS

* INSTANTIATION is what happens when a generic is used. An executable copy of the template is created and actual parameters substituted. An "instance" of the generic is created.

EXCEPTION HANDLERS

KEY ELEMENTS

* Exception Handlers deal with software errors without operator intervention
* Exception events considered
* Execution abandoned
* Handlers may restart under better conditions

EXCEPTION HANDLERS

KEY ELEMENTS

* Allows for user-defined exceptions
* Allows for fault-tolerant programming

DEFINITION

PACKAGE

A package is a group of logically related entities.

PACKAGES

KEY ELEMENTS

* A package forms a collection of logically related entities or computational resources

* A package ENCAPSULATES (puts a wall around these resources)

PACKAGES

KEY ELEMENTS

* Package parts:
  - SPEC: Contact between the implementation and user, identifying visible parts of the package. This interface specifies which parts of the package may be used and how they are used.
  - BODY: implementation hidden from user.

PROS AND CONS OF ADA

PROS

* * Reduces overall life cycle costs
* * Best language tool available to meet the Space Station needs
* * Improves productivity over the life cycle
* * Correctly used, Ada supports software engineering goals and principles

PROS AND CONS OF ADA

CONS

* * Harder to learn
* * Availability of tools and trained personnel
* * Ada environments are not standardized and run time environments are loose

CURRENT STATUS OF ADA

* Increasing number of validated compilers

* Over one billion dollars committed to Ada projects

* Involvement across the government, industrial and academic sectors throughout the free world

CURRENT STATUS OF ADA

* The broadening commitment to Ada is producing a complement of reusable components, libraries of software building blocks and experienced people.

SUMMARY OF KEY POINTS

SOFTWARE
MUST BE DESIGNED
TO WORK CONTINUOUSLY
FOR 15-30 YEARS
AT MINIMUM

2016

AD

SOFTWARE ENGINEERING PRACTICES HOLD PROMISE FOR MEETING LIFE CYCLE NEEDS

SPACE STATION SOFTWARE MUST SATISFY THE SUPER MICE

SUPER MICE

SAFETY
UNDERSTANDABILITY
PORTABILITY
EXTENSIBILITY
RELIABILITY

MODIFYABLE
INTEROPERABILITY
CORRECTNESS
EFFICIENT

ADA WAS DESIGNED TO SUPPORT THE GOALS OF SOFTWARE ENGINEERING

"SOFTWARE ENGINEERING AND THE ROLE OF ADA"

PRESENTATION GLOSSARY

Ada* is a general purpose programming language developed as a result of an international study funded by the United States Department of Defense. The language is ANSI/MIL STD 1815A.
Ada was designed specifically for the domain of large, real-time, embedded computer systems. (G. Booch, pg. xv)
*Ada is a trademark of the US Government, AJPO.

ABSTRACTION is an intellectual tool that allows one to deal with conceptual aspects of a software system apart from the implementation details allowing an overview of an entire system or its components.

ACM - Association for Computing Machinery

AJPO - Ada Joint Program Office

ANSI - American National Standards Institute

ANSI/MIL-STD 1815A-1983 - the approved standards for the Ada programming language.

APSE - Ada Programming Support Environment

CAIS - Common APSE Interface Set

COHESION - how tightly bound or related its internal elements are to one another within a module. (G. Booch, pg. 29)

COMPLETENESS is the process of ensuring that all design elements are present in the system.

CONFIRMABILITY is the evaluation of the software system and its components from a requirements perspective or a design perspective.

COUPLING - a measure of the strength of interconnection among modules. (G. Booch, pg. 29)

CORRECTNESS is the extent to which software is free from design defects and from coding defects - that is fault free - the extent to which software meets its specified requirements and the extent to which software meets user expectations.

DoD - United States Department of Defense

EFFICIENCY is the extent to which software performs its intended functions with a minimum of consumption of computing resources.

EXCEPTION HANDLERS - An exception handler is code that tells the program what to do if an exceptional situation or error occurs.

EXTENSIBILITY is the result of models and rules which allow controlled changes with predictable effects to be made to both interfaces and the models of services and resources on any side of the interfaces.

FACTORING is a process that distributes control so that decisions can be made close to the point at which work is conducted. An example of factoring is the top-down approach to human organizations. (Pressman, pg. 149)

FAN-IN indicates how many modules directly control a given module. (Pressman, pg. 150)

FAN-OUT is a measure of the number of modules that are directly controlled by another module. (Pressman, pg. 150)

GENERICS are parameterized templates of a program unit that allow reuse of code and that allow libraries of programs to be built.

INFORMATION HIDING is the process which removes all unnecessary details from a user's access thereby protecting the software system from unexpected or unwanted changes.

INTEROPERABILITY is the ability to "use" the entities that are "ported" among systems and the properties of the entities, the relationships to other entities, and the properties of these relationships.

KAPSE - Kernel of the Ada Programming Support Environment

LIFE CYCLE - Creation, construction and maintenance of any system from conception to retirement.

LOCALIZATION is the process of creating strongly cohesive programming units, that is, locating elements which exhibit a high degree of functional relatedness within one unit.

MAPSE - Minimal Tool Set of the Ada Programming Support Environment

MIL-STD 1815A-1983 - See ANSI/MIL-STD 1815A-1983

MODIFIABILITY is the ability to control change within software, thus achieving new results without undesirable or disastrous side effects.

MODULARITY is the purposeful structuring of elements (or software modules) that are integrated to satisfy system requirements (loosely coupled).

OBJECT - An object in Ada is any kind of data element, variable or constant.

OBJECT-ORIENTED DESIGN: A methodology that decomposes a problem into separate concerns based upon carefully considered design decisions.

PACKAGE - A package is a group of logically related entities.

PDL - Program Design Language, commonly called pseudocode.

PDR - Preliminary Design Review

PORTABILITY is the ease with which software can be transferred from one computer system or environment to another.

PROJECT OBJECT BASE - source code and software tools available for use.

RELIABILITY is the ability of a program to perform a required function under stated conditions for a stated period of time.

SAFETY is the ability of software to protect life and property in the presence of "N" faults.

"SOFTWARE ENGINEERING is the establishment, and application of sound engineering concepts, principles, models, methods, tools and environments along with standards, guidelines and practices to support computing which is: correct, modifiable, reliable, efficient, and understandable, through the life cycle of the application." -- Charles McKay, 1985

SOFTWARE ENGINEERING METHODS - provide a systematic approach indicating how to develop intermediate software products within the context of the life cycle model.

SOFTWARE ENGINEERING TOOLS - apply automation to software development within the context of the software engineering method.

SOFTWARE LIFE CYCLE - A software engineer's model of the activities and phases involved in the processes of producing and sustaining a system's software products from conception through retirement.

SPECIFICATION - "A specification is a document that prescribes in a complete, precise and verifiable manner the requirements, design, behavior, or other characteristics of a system or system components." (IEEE, 1983)

TASK - A task is a program unit that may execute in parallel with other program units.

TYPING - A type characterizes both a set of values and a set of operations on those values.

UNDERSTANDABILITY is the extent to which the software's algorithms and data structures are easily perceived and easily interpreted.

UNIFORMITY is the degree to which consistent notation is used through the life cycle.

OVERVIEW

Software Engineering and the Role of Ada

Objectives: To introduce the basic terminology and concepts of Software Engineering and Ada. In this seminar the participant will:

* Review the life cycle model.
* Observe the application of the goals and principles of software engineering.
* Gain an introductory understanding of the features of Ada language.

Recommended for: Managers dealing with Shuttle projects, Space Station projects or any software related effort.

Pre-requisites: None

Course Outline:

1. The Software Crisis: Problems and Solutions.
2. The Mandate of the Space Station Program
3. The Software Life Cycle Model
4. Software Engineering
5. Ada Under the Software Engineering Umbrella

Course Material: Notebook

Format: Lecture using foils

Duration: 2 hours

EXECUTIVE SEMINAR
OUTLINE

Software Engineering and the Role of Ada

I. The Software Crisis: Problems and Solutions

Discuss the "software crisis" environment. Identify the key elements and causes of this crisis.

* Over budget and late
* Actual life cycle cost
* Modification is difficult, time consuming and costly
* The Software Invasion

II. Mandate of the Space Station Program

1.0 Provide a brief profile of the Space Station program:

* Large
* Complex
* Distributed networks
* Embedded components
* Long-term life expectancy
* Non-stop operation
* Over 100 million lines of code

2.0 Describe the software challenge imposed by such projects:

* Many needs initially undetermined and unknown
* Many requirements initially undefined
* Personnel continuity an unrealistic goal
* Vendor continuity an unrealistic goal
* Many needs are never fully determined - always changing
* Integration of new functions in an incrementally evolving system

3.0 Identify the software requirements imposed by this challenge.

* MODIFIABILITY
* EFFICIENCY
* RELIABILITY/SAFETY
* UNDERSTANDABILITY
* CORRECTNESS
* PORTABILITY/INTEROPERABILITY/EXTENSIBILITY

III. The Software Life Cycle

1.0 Define Software Life Cycle.

2.0 Briefly discuss the components of the life cycle model pertinent to NASA/JSC projects

2.1 Acquisition Activities
NASA Software Acquisition Life Cycle Model.
(Software Management & Assurance Program)

* Software Concept & Project Definition
* Software Initiation
* Software Requirements Definition
* Software Architecture Design
* Software Detail Design
* Software Implementation
* Software Systems Integration & Testing
* Software Acceptance Testing & Delivery
* Operation & Maintenance Transition

2.2 Sustaining Engineering Activities

* System Requirements Analysis
* Software Requirements Analysis
* Preliminary Design
* Detailed Design
* Coding and Unit Test
* Computer Software Component Integration

2.3 Supporting Activites

* Documentation
* Configuration Management
* Quality Management
* Review
* Verification & Validation
* Automated Support
* Communication through the Project Object Base

3.0 Discuss the impact of change in terms of time and money across the life cycle. Include a specific reference to the current cost of the sustaining engineering or maintenance phase. Suggested solutions:

* Early error detection
* Reusable componenets
* High quality documentation
* Automated tools and methods

IV. Software Engineering

1.0 Give working definitions of Software Engineering.

1.1 Discuss the move from traditional Computer Science to Software Engineering in industry and academia.

1.2 Give examples of projects experiencing productivity increases and fewer errors with the application of software engineering methods.

2.0 The goals of software engineering:

* MODIFIABILITY
* EFFICIENCY
* RELIABILITY
* UNDERSTANDABILITY
* CORRECTNESS

3.0 Identify and briefly describe the principles of software engineering.

* ABSTRACTION
* INFORMATION HIDING
* MODULARITY
* LOCALIZATION
* CONFIRMABILITY
* COMPLETENESS
* UNIFORMITY

4.0 Briefly identify some of the tools and methods for applying software engineering goals and principles to the life cycle phases. For each of the tools and methods discussed give a brief background, when it was developed, by whom, and who uses it.

Methods:

* Structured Analysis & Design Techniques (SADT)
* Structured Design
* Jackson's Data Flow Design
* Object Oriented Design (OOD)

Tools

* Program Design Language (PDL)
* Other Tools
  - Languages
  - Editors
  - File Managers

V. Ada Under a Software Engineering Umbrella

1.0 Introduce Ada language and environments with a brief historical overview, identifying the milestones of its development.

1.1 Give the rationale for its development, the DOD study and findings. Identify the design teams in the competition, the winning team and team leaders (Jean Ichbiah, J.G.F. Barnes, R. Firth). Explain how Ada was named.

1.2 Review the milestones in the evolution of requirements for Ada programming environments. Give a synopsis of the Stoneman architecture suggested for these environments, including KAPSE, MAPSE, APSE. Discuss the development of reusable, sharable libraries of tools. Be explicit about the current availability of these tools.

2.0 Briefly identify the unique Ada features and its relationship to software engineering.

* STRONG SPECIFICATION
* STRONG TYPING
* TASKS
* GENERICS
* EXCEPTION HANDLERS
* PACKAGES

3.0 Summarize the prose and cons of Ada.

PROS:

* Reduces overall life cycle costs
* Best procedural language tool available to meet Space Station needs
* Improves productivity over the life cycle
* Correctly used, Ada supports software engineering goals and principles

CONS:

* Harder to learn
* Availability of tools and trained personnel
* Ada environments are not standardized and run time environments are loose

4.0 Summarize the current status of Ada

* Increasing number of validated compilers.
* Over one billion dollars committed to Ada projects.
* Involvement across the military, industrial and academic sectors.
* Production of reusable components; building of libraries.

5.0 Conclude the session by summarizing the points covered.

* Overall life cycle costs must be reduced.
* New approaches are needed to meet the software challenge of the future and growing life cycle issues.
* It is imperative to identify sound software engineering strategies.
* Software engineering techniques must be applied across the life cycle.
* Ada, baselined for the Space Station, was designed to implement the goals and principles of software engineering.