

# **Source Code Reuse Mechanism Project Outline**

December 2, 2002

**Aiyana Brooks**

**Juan Castro**

**Gregory Horvath**

**Sandra Martinez**

**Noel Villegas**

Technical Advisor

Professor John C. Kiss

## Our Approach to Creating a Code Reuse Mechanism

- I. Generative Programming based Design/Policy Based Design – design software system families that allow the creation of highly customized and optimized intermediate or end-products, automatically manufactured on demand from elementary, reusable implementation components.
  - a. Features of GP/Policy Based Design
    - i. Assembly line form of software engineering as compared to the ‘cutting filing of car parts in order to assemble a car’ way of component based software engineering.
    - ii. Captures as much production knowledge in program form as possible. In order to ‘know how you got there’ in the software development process. Includes configuration knowledge, testing strategies, error diagnosis. Packaged as reusable libraries, these are known as active libraries.
    - iii. Creation of maximally reusable components that can be interchanged, composed, and customized at user will.
  - b. GP composed of
    - i. Domain Engineering– focuses on system families to develop reusable libraries.
      1. Domain analysis
        - a. Domain definition
          - i. Domain scoping
      2. Domain modeling
        - a. Feature modeling
      3. Domain design
      4. Domain implementation
    - ii. Generic Programming – programming with abstract types. Realized as templates in C++.
    - iii. Generators - assembly of implementation components. We will implement with
      1. Template Metaprogramming in C++
        - a. code generation
        - b. type detection
        - c. compile time programs

- II. Standardized In Code Documentation Format
  - a. File Comment Header Block
    - i. File Name
    - ii. Description
    - iii. Author / Revision History
    - iv. Dependencies
  - b. Function Comment Header Block
    - i. Function Name
    - ii. Description
    - iii. Inputs
    - iv. Outputs
    - v. Pre-conditions
    - vi. Post-conditions
    - vii. Invariants
    - viii. Return value

- III. Create a Reuse Document for each library containing
  - a. Feature Models - (design level functionality) abstract, concise, explicit representation of the variability present in software
    - i. Features or Concepts
    - ii. Alternative Features/Concepts
    - iii. Sub Features (extension points)
    - iv. Semantic Description
    - v. Rationale for Features
    - vi. Stakeholders
    - vii. Exemplar Systems
    - viii. Constraints
  - b. UML Models (implementation level relationships)
    - i. Document interaction
    - ii. Approach the system through a small set of nearly independent views of a model [7]
    - iii. Define several (if not all) of the following
      - 1. Use case diagrams
      - 2. Class diagrams
      - 3. Behavior diagrams
        - a. State chart
        - b. Activity diagram
        - c. Interaction diagrams
          - i. Sequence diagrams
          - ii. Collaboration diagrams
        - d. Implementation diagrams
          - i. Component diagram
          - ii. Deployment diagram
  - c. Code Flowcharts - document code flow for classes, functions
    - i. Entry points
    - ii. Exit points
    - iii. Returns
    - iv. Thrown exceptions
      - 1. caught exceptions
      - 2. uncaught exceptions
    - v. Critical Sections
    - vi. Calls
  - d. Reuse Guidelines
    - i. Correctness - the ability of software to perform exact tasks as defined by their specification. Related to error propagation and accuracy.
    - ii. Robustness - the ability of software systems to react appropriately to abnormal conditions.
    - iii. Performance/Efficiency - the ability of a software system to place as few demands as possible on hardware resources such as processor time, internal and external memories, and communication bandwidth

1. algorithmic complexity of performance
  2. algorithmic complexity of memory (stack, heap) usage
- iv. Extensibility - the ease of adapting software to changes in specification
  - v. Compatibility - the ease of combining various unassociated software elements with others
  - vi. Portability - the ease of transferring software products to various hardware and software environments