Klocwork Architecture Excavation Methodology

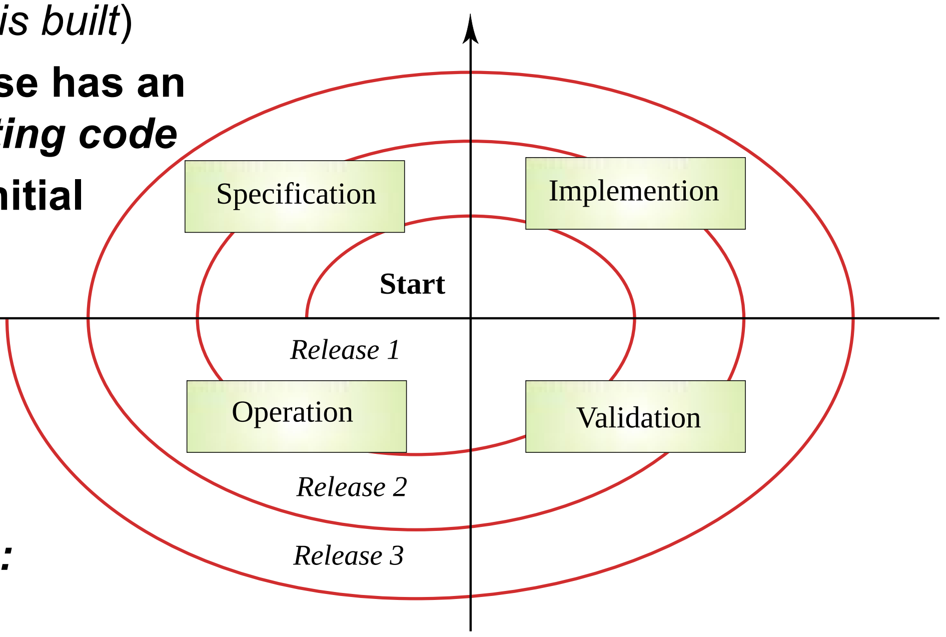
Nikolai Mansurov
Chief Scientist & Architect

Overview

- **Introduction**
  - Production of software is evolutionary and involves multiple releases
  - Evolution of existing code has significant economic impact
  - Why the cost of production increases over time ?
- **Architecture Excavation : single most important investment in software production management**
  - Challenges
- **Klocwork Architecture Excavation Methodology**
  - Focus
  - Container Models
  - Basic Operations
  - Patterns
  - Strategy
- **Examples**
- **What next?**
- **Key points**

Production of software is evolutionary

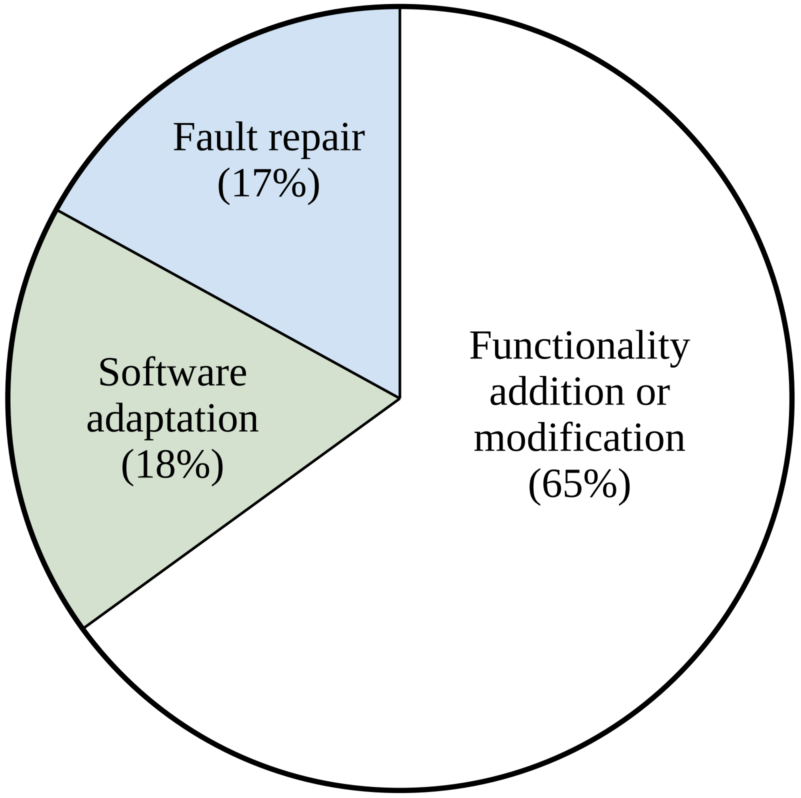
- **Production of software involves multiple releases**
  - (despite the fact that many textbooks emphasize only the *initial* release, when *the system is built*)
- **Software production after the initial release has an additional *constraint* of dealing with *existing code***
- **Changes are made to software after the initial release in order to:**
  - Develop new functionality and features
  - Fix bugs
  - Adapt to new operating environments
  - Improve quality
- **These activities have different *magnitude:***
  - *Maintenance (repair)*
  - *Major new features to existing software*
  - *Modernization*
  - *(Redevelopment)*
- **Activities of *increasing magnitude* are required to address changing business needs**



From Ian Sommerville, Software Engineering, 2000

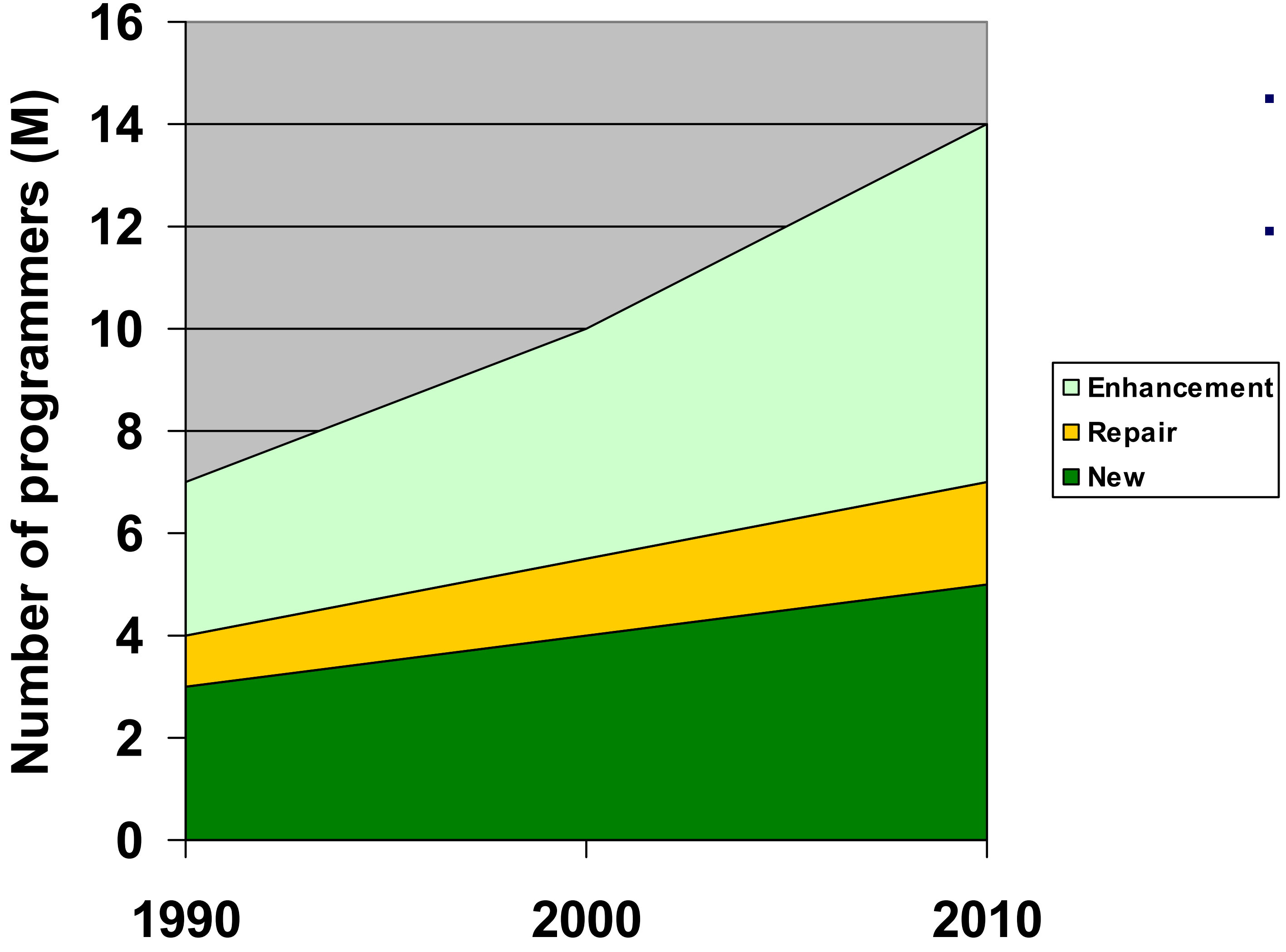
Software production with existing code

- **Maintenance**
  - Corrective maintenance
  - Adaptive maintenance
  - Perfective maintenance
  - Preventive maintenance
- **Major new features to existing software**
  - Major modifications
  - Scaling
- **Modernization (beyond maintenance)**
  - Porting to a new platform
  - Migration to a new technology
  - Migration to COTS components
  - Modularization and refactoring
- **Redevelopment**



From Ian Sommerville, Software Engineering, 2000

Evolution of existing code has significant economic impact

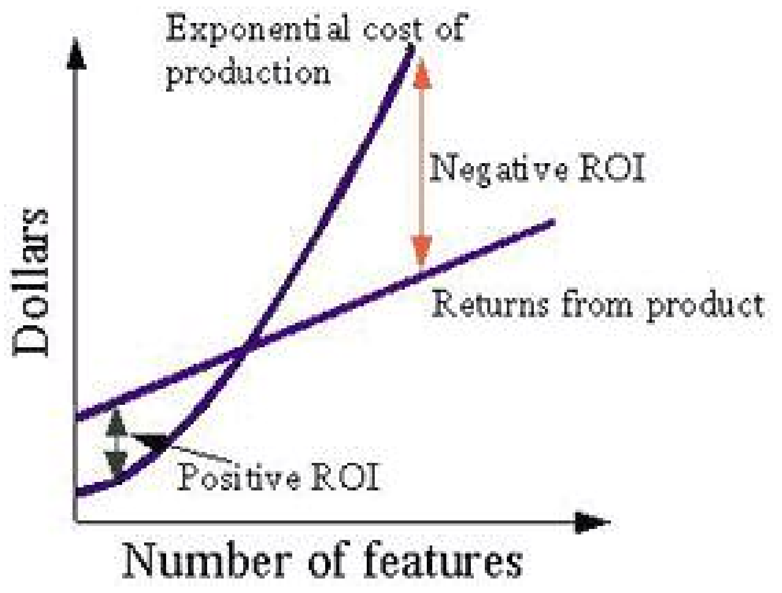


From Deursen,Klint,Verhoef,1999

- **More than one half of all programmers are *already* working with existing code (repair + enhancement)**
- **The cost of post-initial evolution has grown from 40% to 90% of the total production cost**
- **The number of programmers working with existing code grows faster than the number of programmers developing new software**
  - Larger amounts of software already developed
  - Large business value accumulates in existing code over time
  - Increasing cost of new development
  - The useful lifespan is increasing
  - Bigger churn of platforms and technologies
  - More defects

Why the cost of production increases over time ?

- **The cost of changing software increases over time, and is not linear, but exponential**
  - Brooks attributes the exponential rise in costs to the cost of communication.
  - Maintenance corrupts the software structure so makes further maintenance more difficult
  - The number of error reports increases
  - Productivity of the maintenance staff decreases
  - Code becomes harder to understand
- ***Architecture Erosion*: Positive feedback between**
  - the loss of software architecture coherence
  - the loss of the software knowledge
    - less coherent architecture requires more extensive knowledge
    - if the knowledge is lost, the changes will lead to a faster deterioration
- **loss of key personnel = loss of knowledge**
- **Exponential cost of production may lead to *project failures* (Negative ROI)**



From C.Mangione, CIO Update, 2003

Architecture Excavation

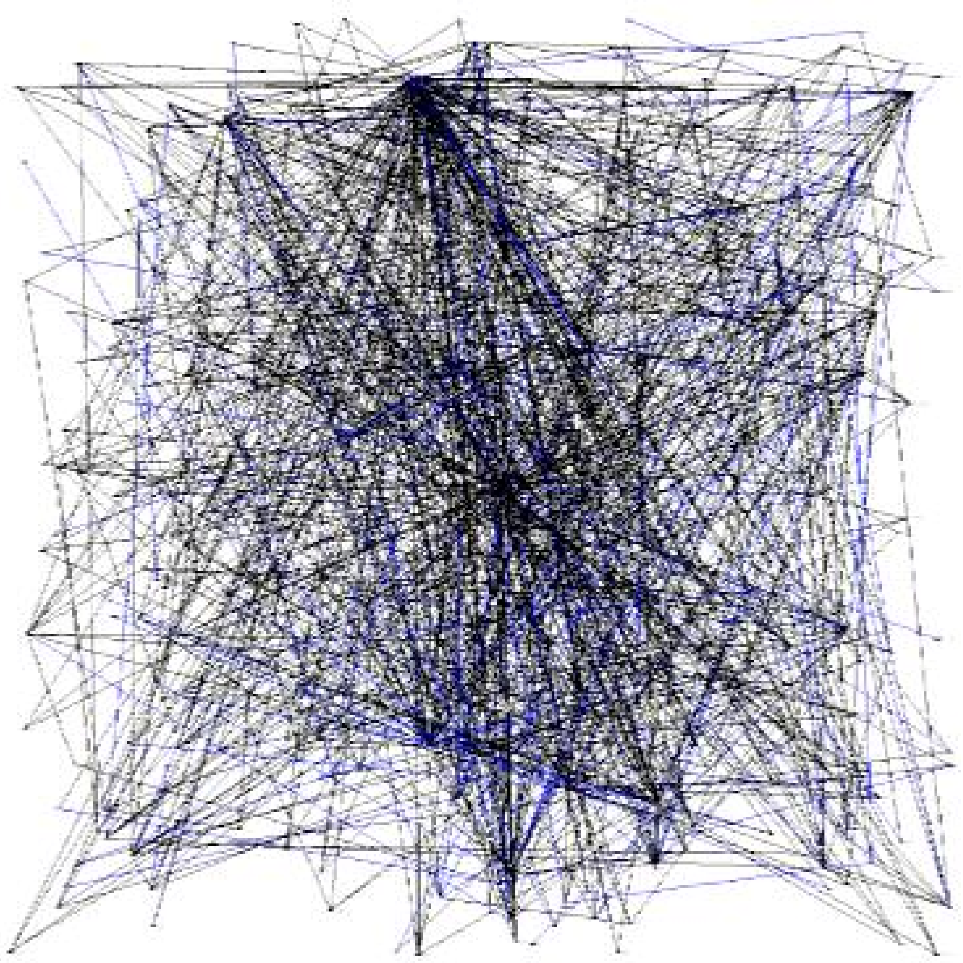
- **Visualize *Architecture* of the "as built" software as a *model***
- **"*Excavation*", because the model is likely to be created manually by abstracting from existing code**
  - Abstraction level is *sufficient*, when the architect can
    - Explain the model to the team
    - Use the architecture model to detect anomalies
    - Plan refactorings through the model
    - Enforce integrity of the model

- **Preserve and *automatically update* the model as changes are made to the software – helps preserve and accumulate knowledge at minimum effort**
- **Align the "as built" model with the "as designed" description, if available**
- **Use the architecture model to proactively refactor software**
  - Minor refactorings to restore architecture cohesion
  - Larger refactorings to improve architecture robustness
  - Major refactorings (modernizations, reuse, redevelopment)
- **Use the architecture model to enforce integrity of the software during the evolution**

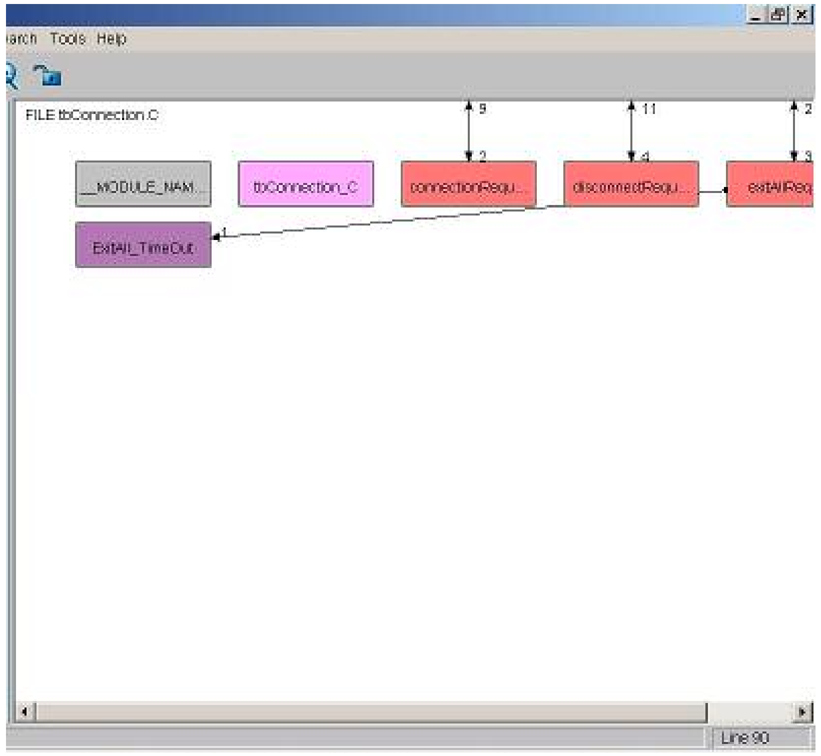
What are the challenges of Architecture Excavation?

- Comprehension issues
- Architecture is intangible
- Gap between code and architecture levels
- Erosion makes it more difficult to recover original vision
- There exist multiple architecture views/concerns
- The model should be used at multiple abstraction levels
- Scope vs precision
- Distributed design plans

What’s wrong with low levels of abstraction?



..either too many model elements



..or insufficient scope

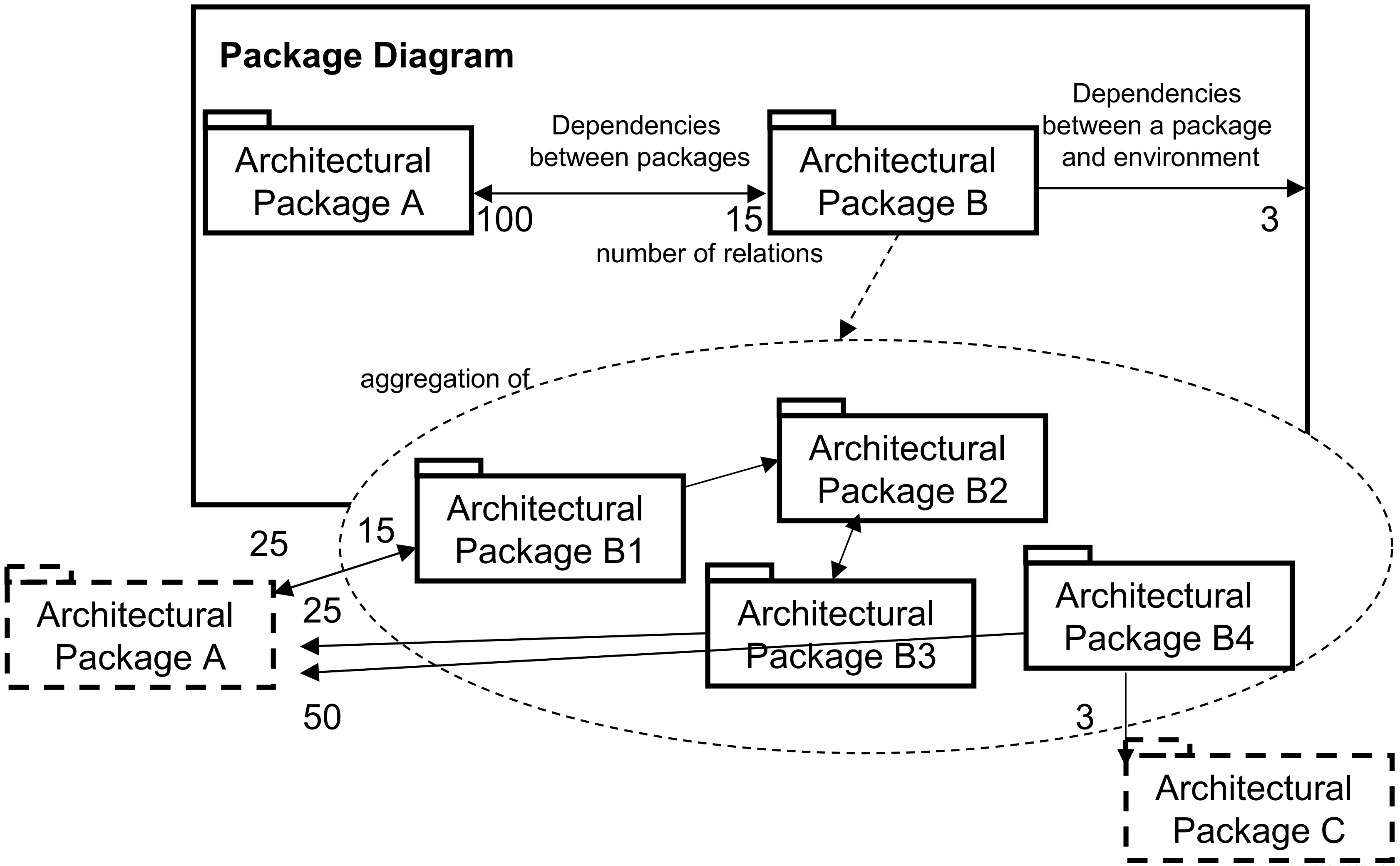
Klocwork Architecture Excavation Methodology

- **Focus:**
  - Containers, interfaces, dependencies on top of entities and relationships
- **Container Models**
  - Are scalable and precise; can be used for both abstraction and refactoring
  - Not UML, because of scalability, the need to evolve with the code, and specific "existing code" understanding concerns, like links to the code, navigation, etc.
  - Transition to UML is straightforward
- **Strategy**
  - Top-down
  - As deep as necessary, as shallow as possible
  - incremental
- **Operations:**
  - Aggregation of entities into bigger containers
  - Refactoring (moving entities between containers)

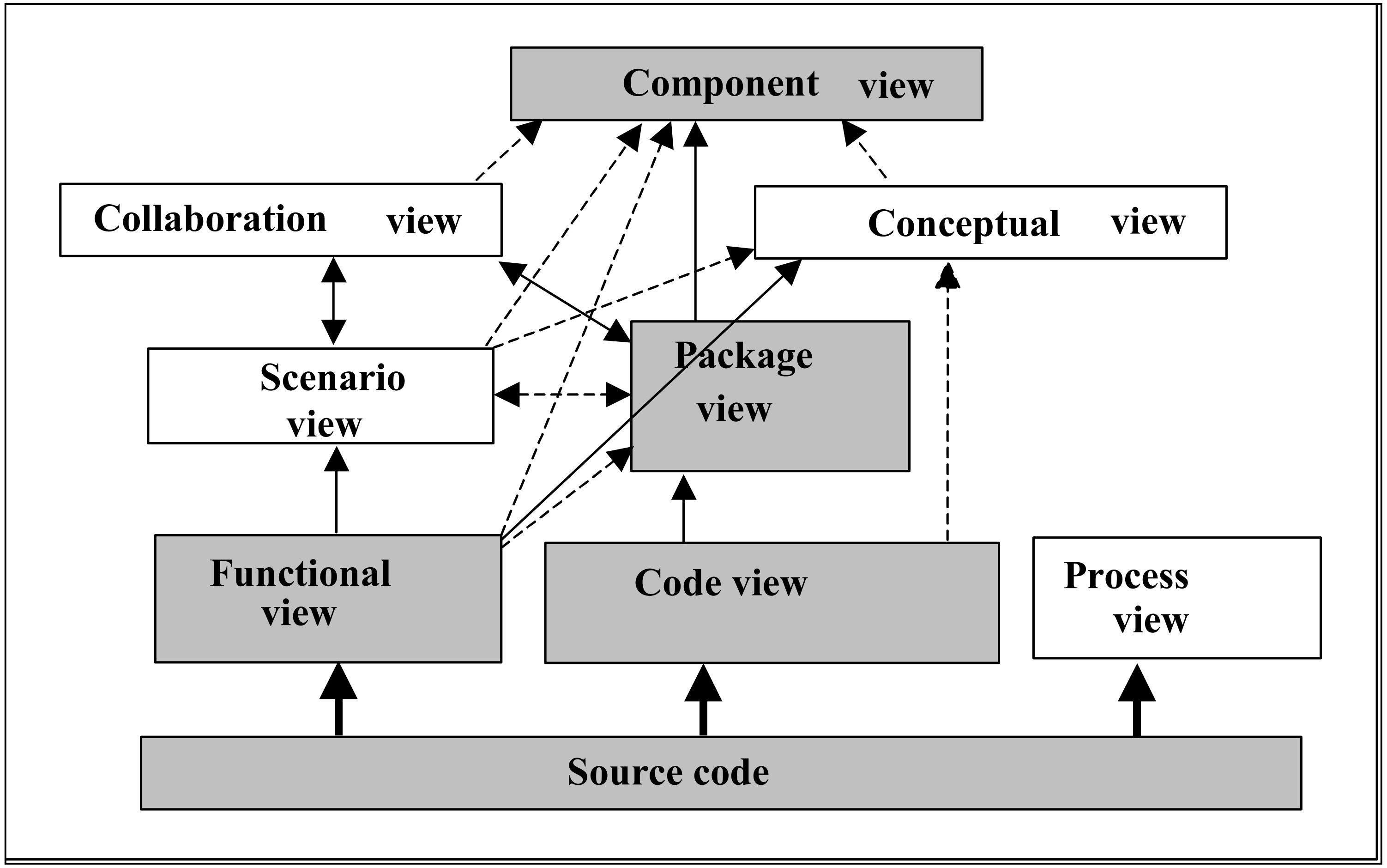
Container models

- **Represent "containers" and relations between "containers":**
  - each "container" has dependencies on other "containers"
  - each "container" provides an API to other "containers"
- **This model is scalable:**
  - aggregation of "containers" is another "container"
  - The aggregation depends on everything that individual parts depended on
  - The aggregation provides the union of APIs, provided by individual parts
- **Model can be refactored**
  - subcontainers can be moved from one container to another, model shows how dependencies and APIs change
- **This model is precise**
  - With respect to the contents of aggregations
  - With respect to APIs
- **This model is meaningful and useful**
  - Leaf "containers" can be procedures, variables, files, etc.
  - Leaf relations (APIs) are e.g. procedure-calls-procedure, etc.
- **Model can be preserved and automatically updated as changes are made to software**
  - Leaf containers and their relations are automatically extracted from source; the model stores only the hierarchy of containers; relations are recalculated on-the-fly

Container models



Containers can represent multiple architecture views



Excavate logical architecture: overview

The excavation process consists of the following 5 phases:

1. **Collect existing architecture information**

   During this phase objectives for the excavation are reviewed, existing architecture information is collected and reviewed, initial architecture view of the software is reviewed, target architecture view is selected

2. **Identify components**

   During this phase we identify subsystems that belong to same high-level component and group them together. The newly recovered component is given a meaningful name, is meaningfully resized and is meaningfully placed on the diagram

3. **Evaluate coupling and refactor**

   During this phase coupling between high-level components is evaluated. Use refactoring in order to simplify dependencies between components. Move the anomalous subcomponent, even as small as an individual symbol, to a different container, and reduce the coupling.

4. **Evaluate cohesion and refactor**

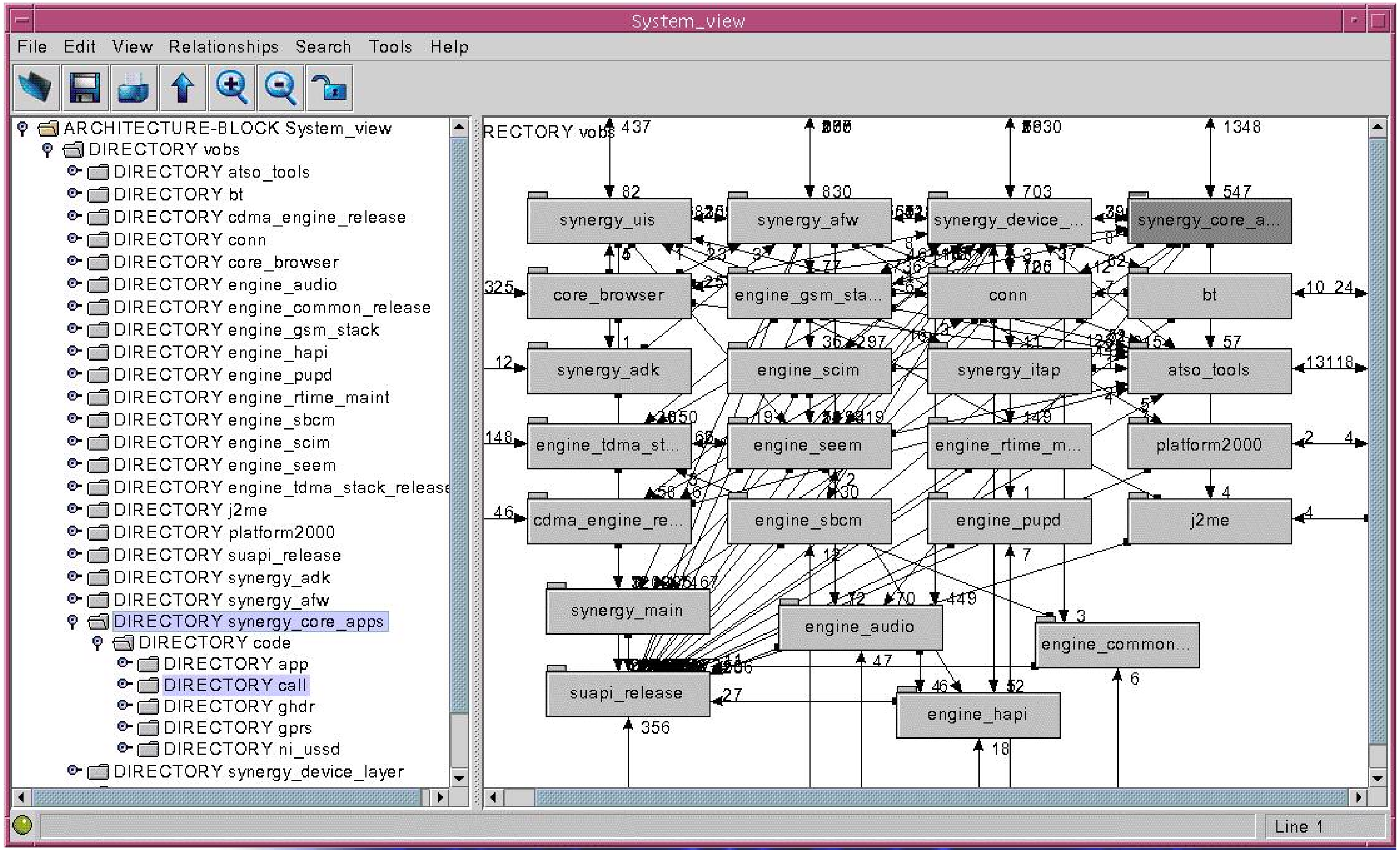
   During this phase cohesion of selected components can be evaluated. Use refactoring In order to improve cohesion of components. Identify cohesion anomalies and move to proper containers or collect in new components

   Phases 2,3 and 4 can be repeated iteratively (recovering more high-level components after refactoring has sufficiently simplified the view) as well as recursively (excavating subcomponents of high-level components) , based on the initial objectives

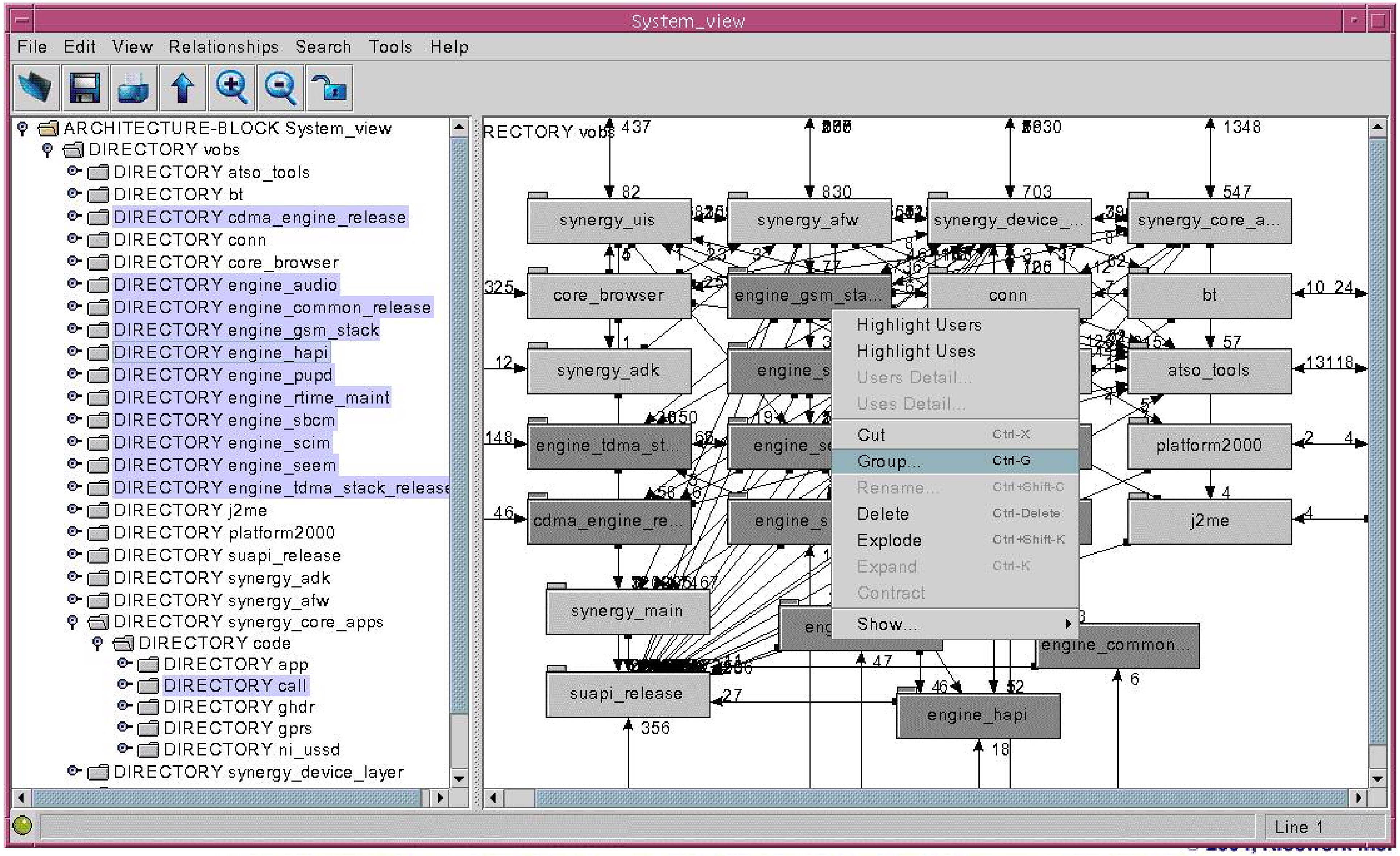
5. **Finalize architecture description**

   Excavated architecture model should be saved and exported as XML file to be used fo subsequent software builds. Thus the excavated architecture model becomes a “living” document.

Excavation: Initial component model

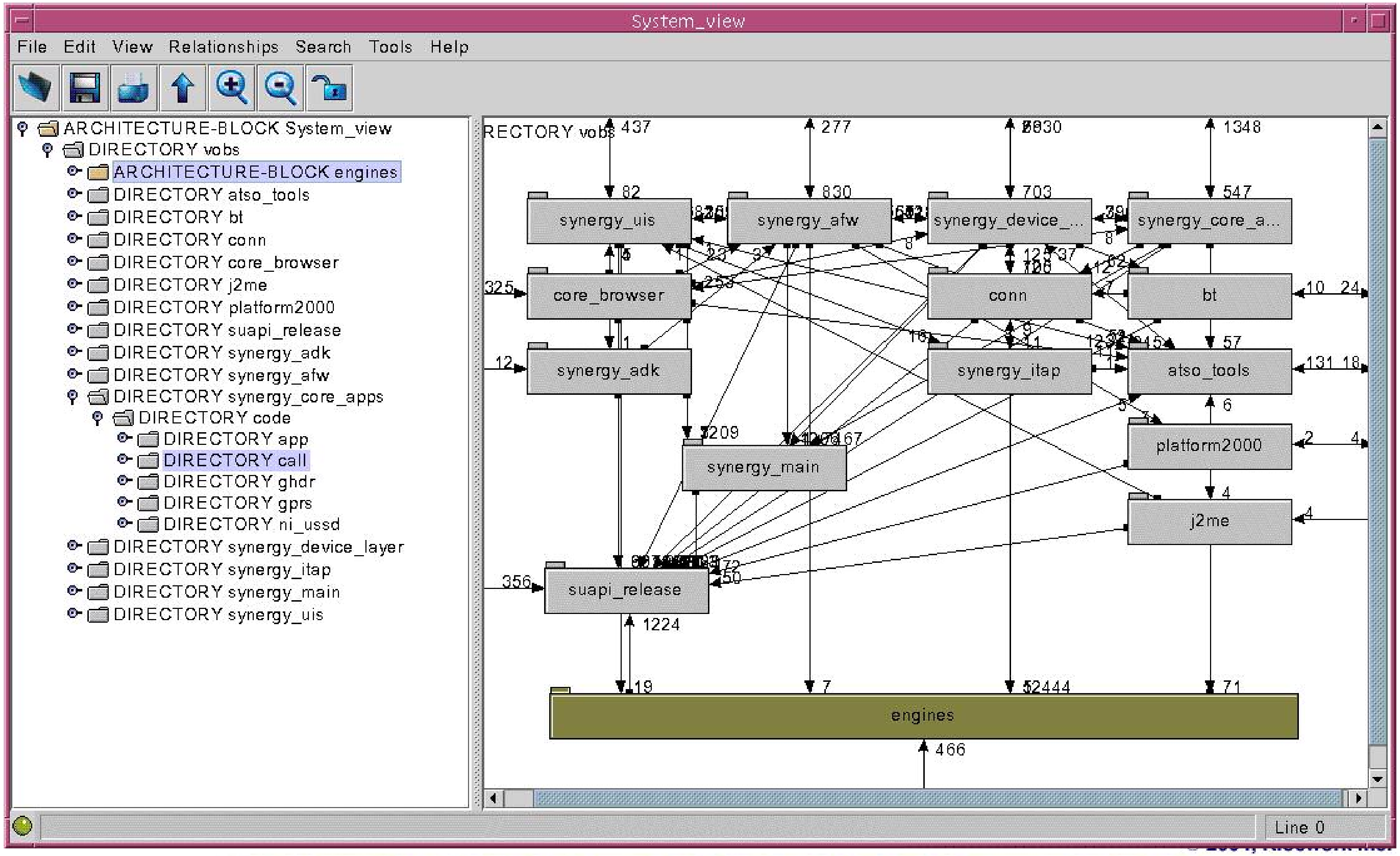


Excavation: Identifying components



```
ARCHITECTURE-BLOCK System_view
  DIRECTORY vobs
    DIRECTORY atso_tools
    DIRECTORY bt
    DIRECTORY cdma_engine_release
    DIRECTORY conn
    DIRECTORY core_browser
    DIRECTORY engine_audio
    DIRECTORY engine_common_release
    DIRECTORY engine_gsm_stack
    DIRECTORY engine_hapi
    DIRECTORY engine_pupd
    DIRECTORY engine_rtime_maint
    DIRECTORY engine_sbcm
    DIRECTORY engine_scim
    DIRECTORY engine_seem
    DIRECTORY engine_tdma_stack_release
    DIRECTORY j2me
    DIRECTORY platform2000
    DIRECTORY suapi_release
    DIRECTORY synergy_adk
    DIRECTORY synergy_afw
    DIRECTORY synergy_core_apps
      DIRECTORY code
        DIRECTORY app
        DIRECTORY call
        DIRECTORY ghdr
        DIRECTORY gprs
        DIRECTORY ni_ussd
    DIRECTORY synergy_device_layer
```

Excavation: Aggregating components

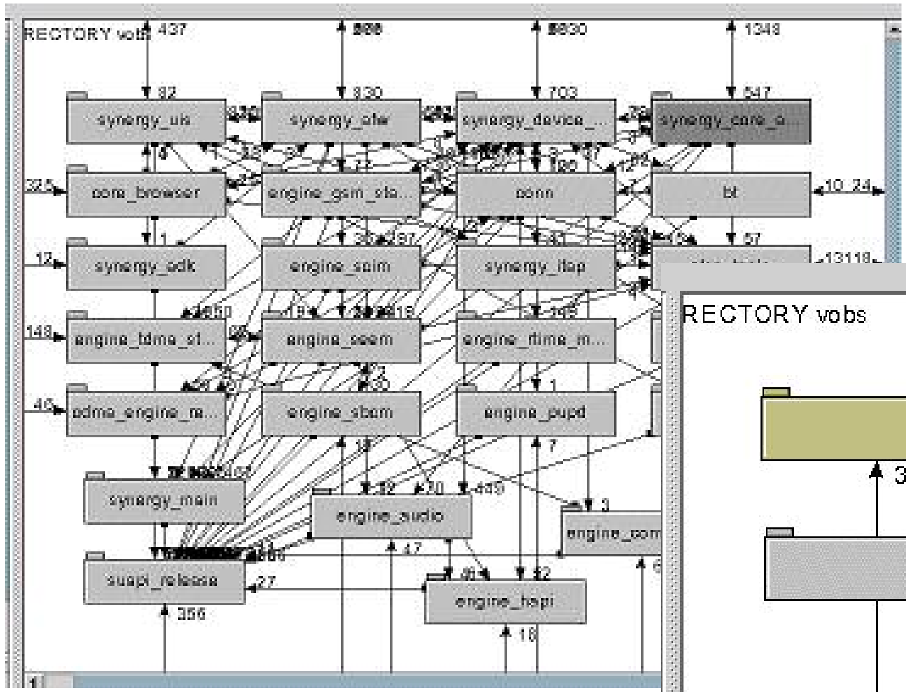


System_view

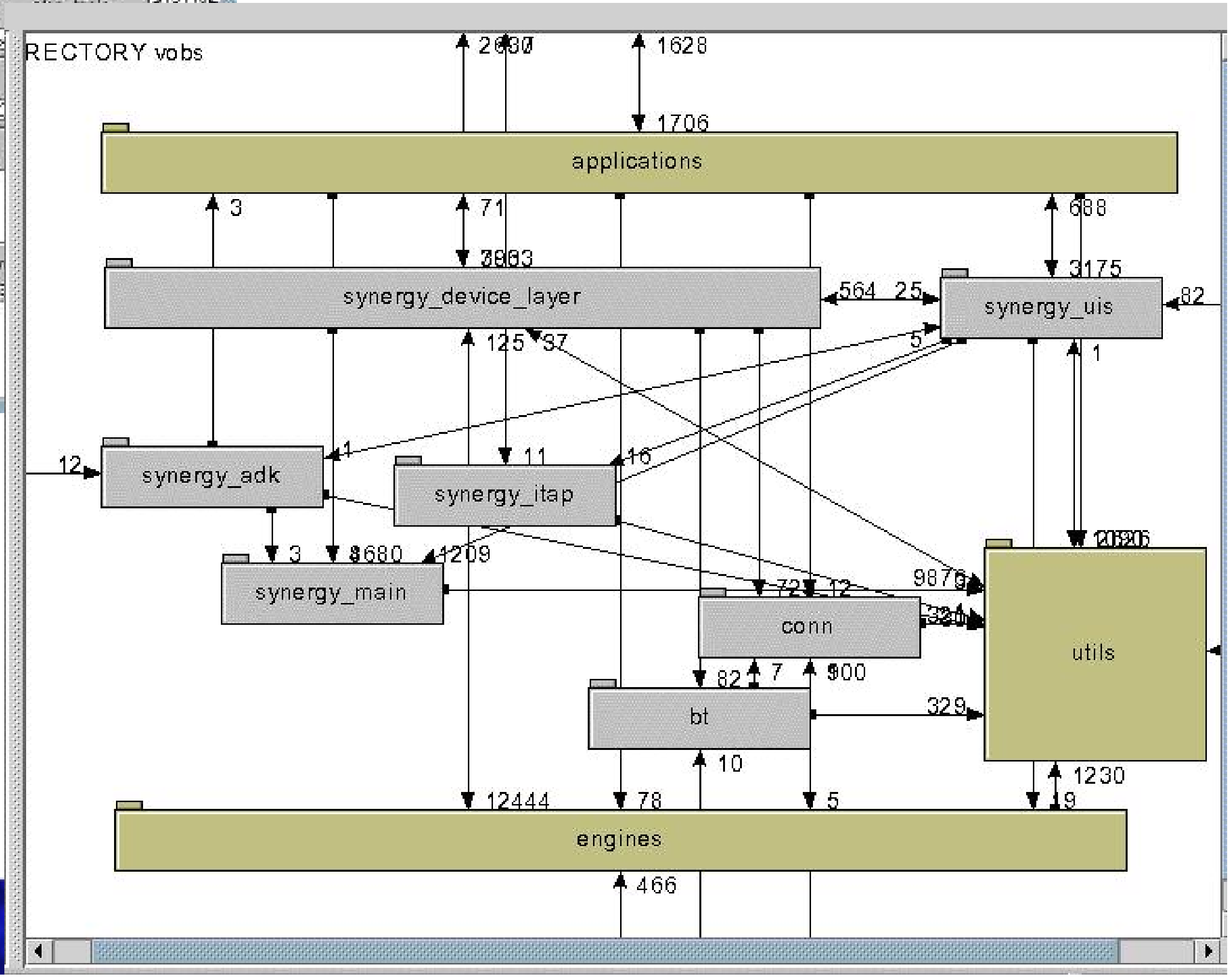
File Edit View Relationships Search Tools Help

- ARCHITECTURE-BLOCK System_view
  - DIRECTORY vobs
    - ARCHITECTURE-BLOCK engines
    - DIRECTORY atso_tools
    - DIRECTORY bt
    - DIRECTORY conn
    - DIRECTORY core_browser
    - DIRECTORY j2me
    - DIRECTORY platform2000
    - DIRECTORY suapi_release
    - DIRECTORY synergy_adk
    - DIRECTORY synergy_afw
    - DIRECTORY synergy_core_apps
      - DIRECTORY code
        - DIRECTORY app
        - DIRECTORY call
        - DIRECTORY ghdr
        - DIRECTORY gprs
        - DIRECTORY ni_ussd
    - DIRECTORY synergy_device_layer
    - DIRECTORY synergy_itap
    - DIRECTORY synergy_main
    - DIRECTORY synergy_uis

Excavated high-level architecture



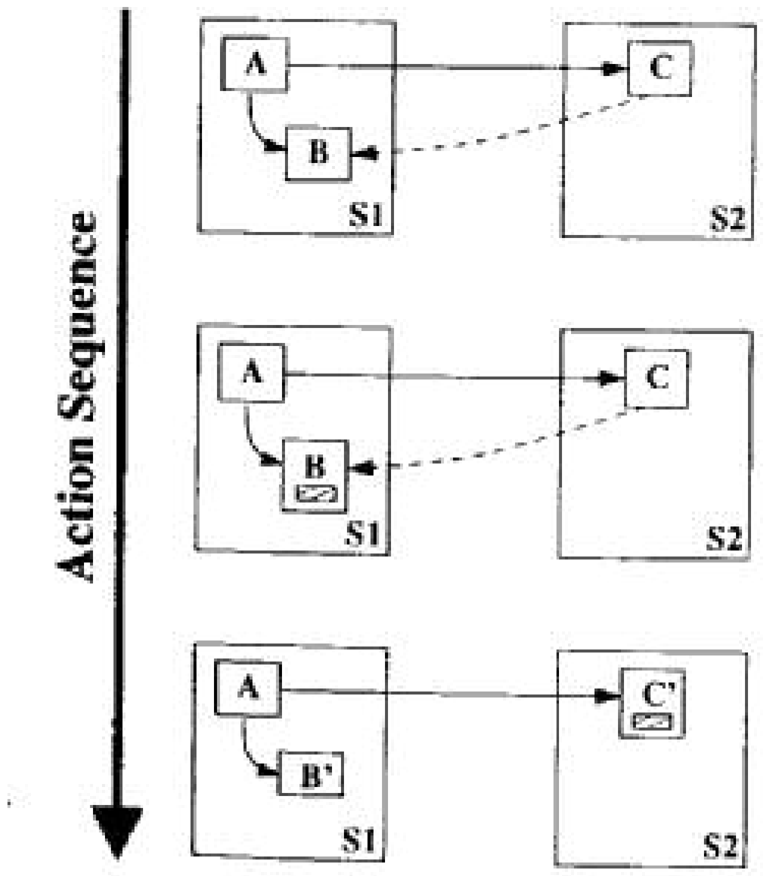
before



after

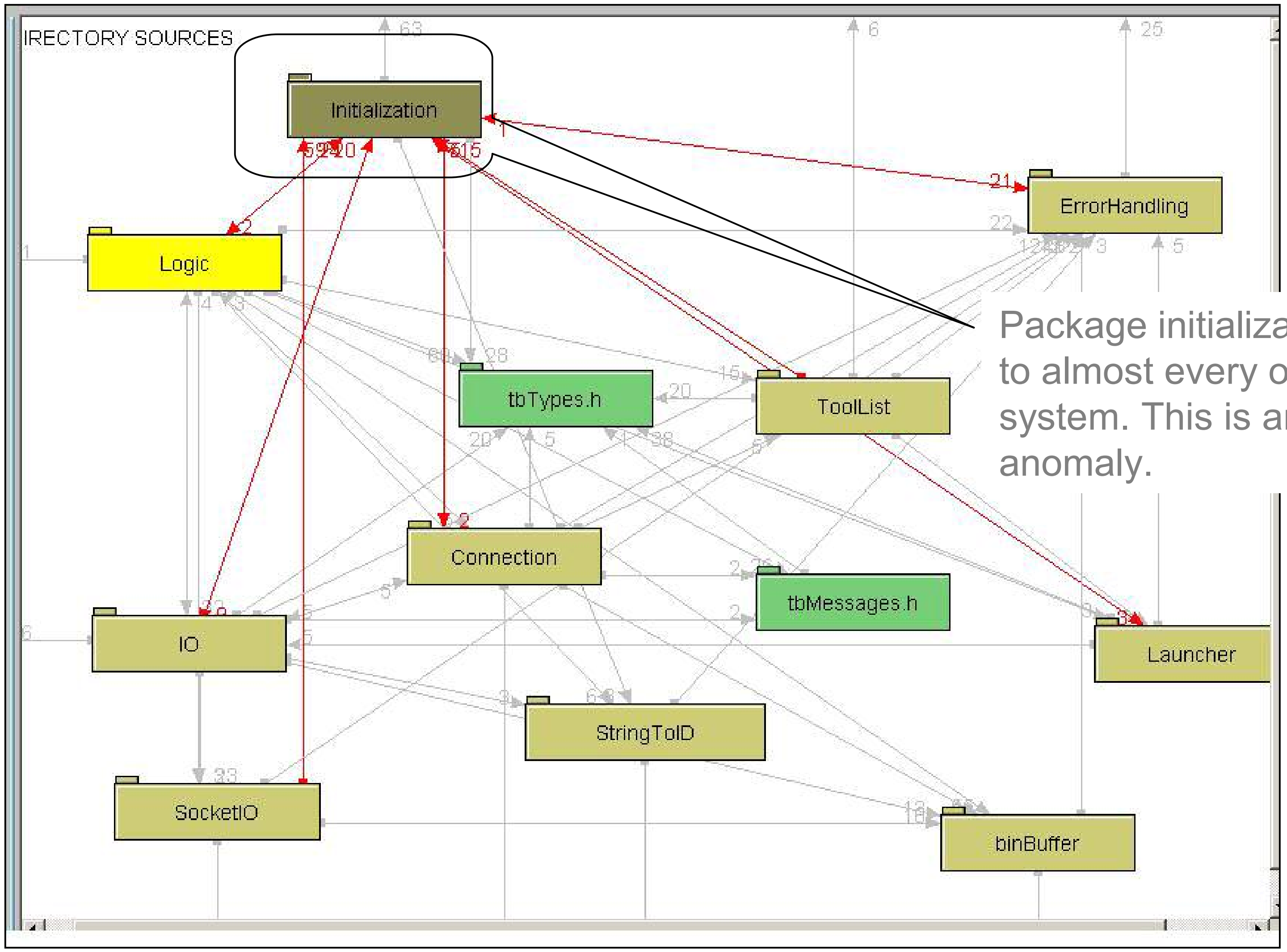
Klocwork

Refactoring removes accidental coupling



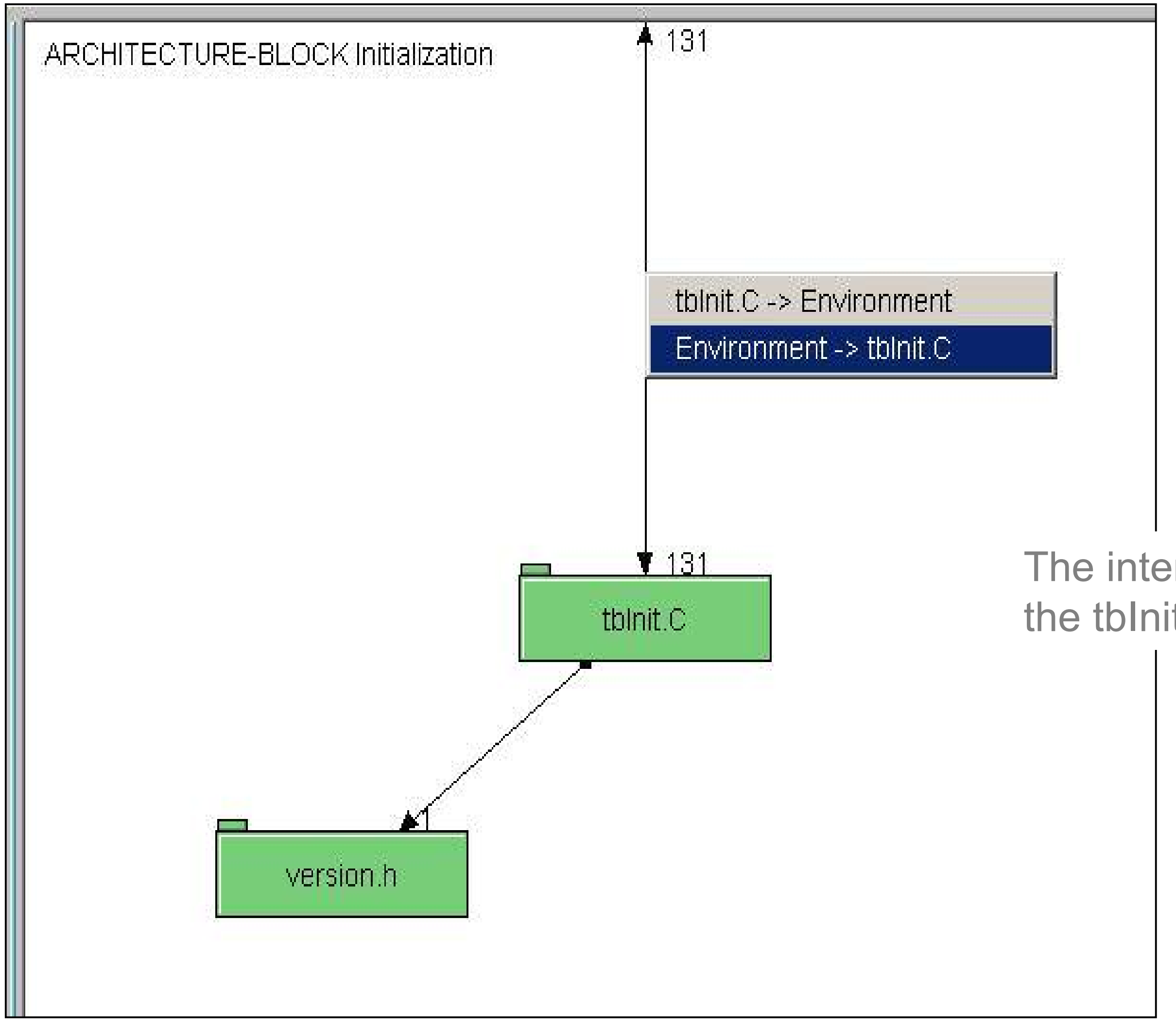
From R.Holt, ICSM, 200

Refactoring (1/6)



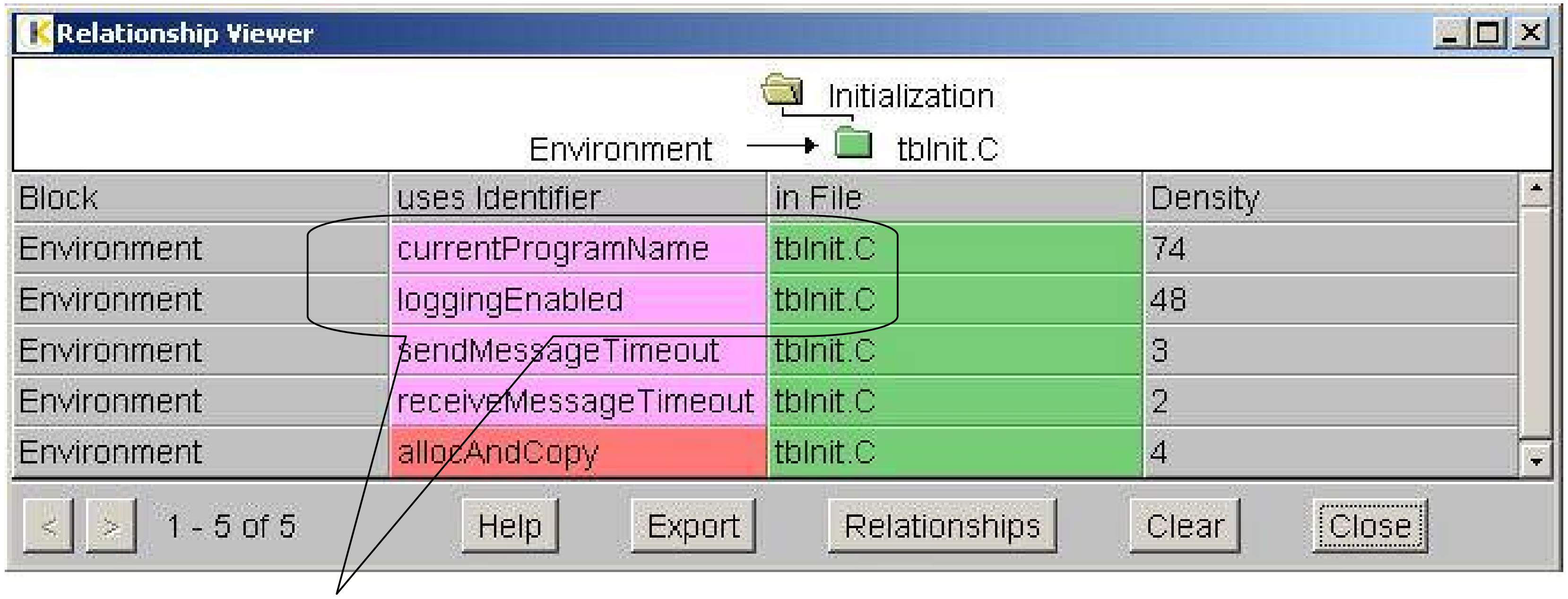
Package initialization provides interface to almost every other package in the system. This is an architectural anomaly.

Refactoring (2/6)



The interface is provided by the tbInit.C file

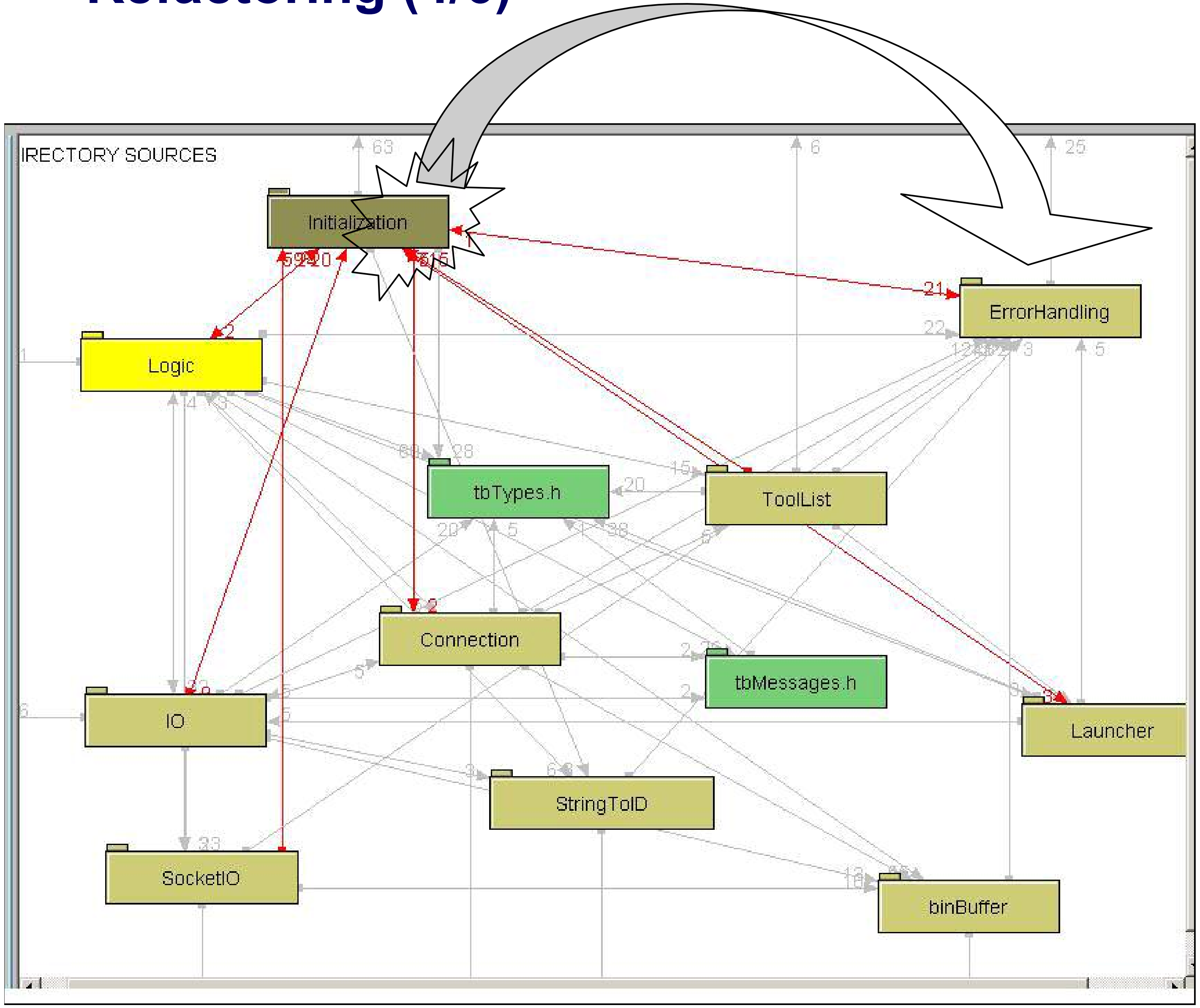
Refactoring (3/6)



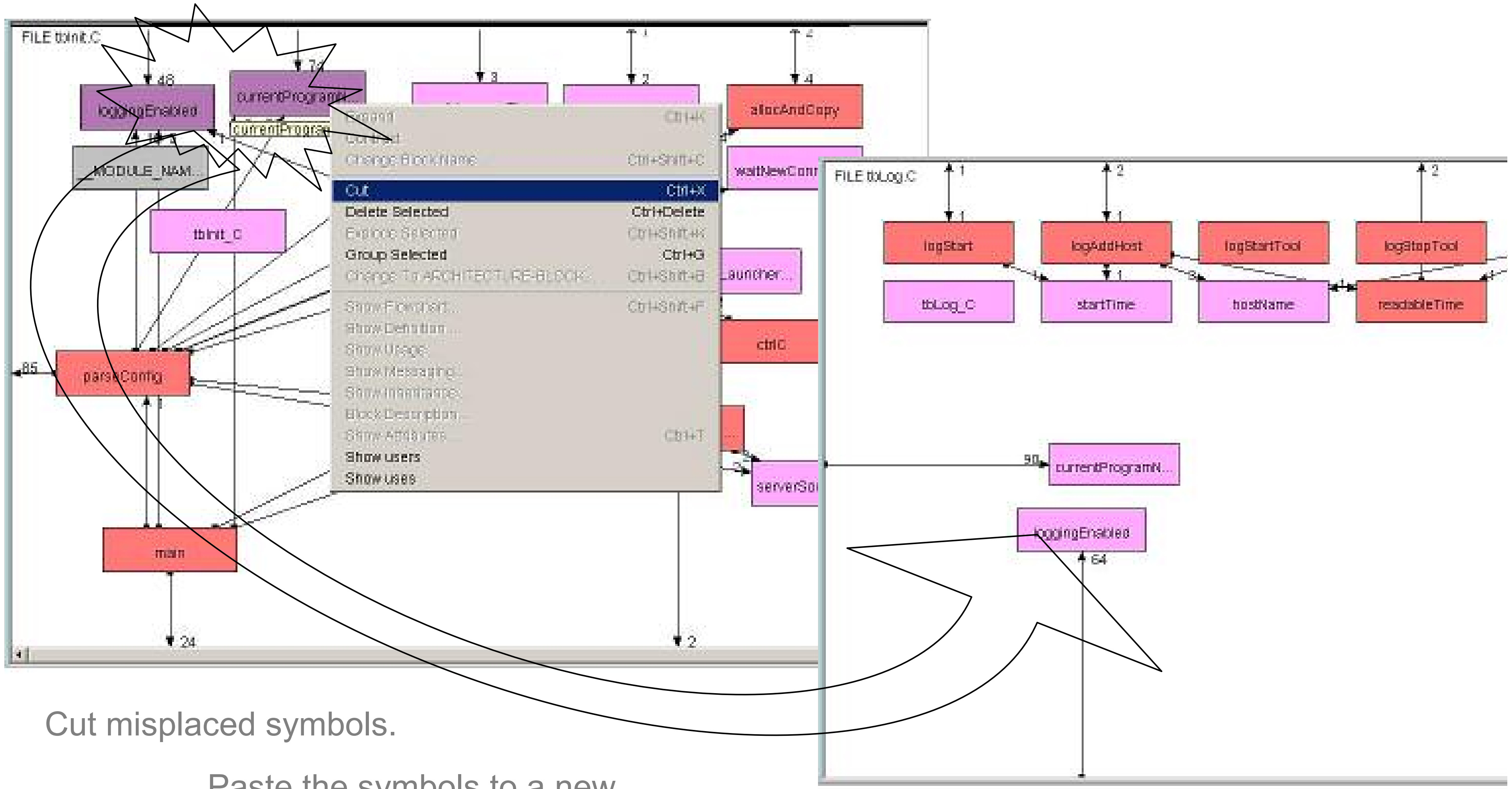
Block	uses Identifier	in File	Density
Environment	currentProgramName	tbInit.C	74
Environment	loggingEnabled	tbInit.C	48
Environment	sendMessageTimeout	tbInit.C	3
Environment	receiveMessageTimeout	tbInit.C	2
Environment	allocAndCopy	tbInit.C	4

The interface consists of only 5 symbols. Variables **CurrentProgramName** and **loggingEnabled** are very frequent. Look For comments to find that they belong to logging.

Refactoring (4/6)



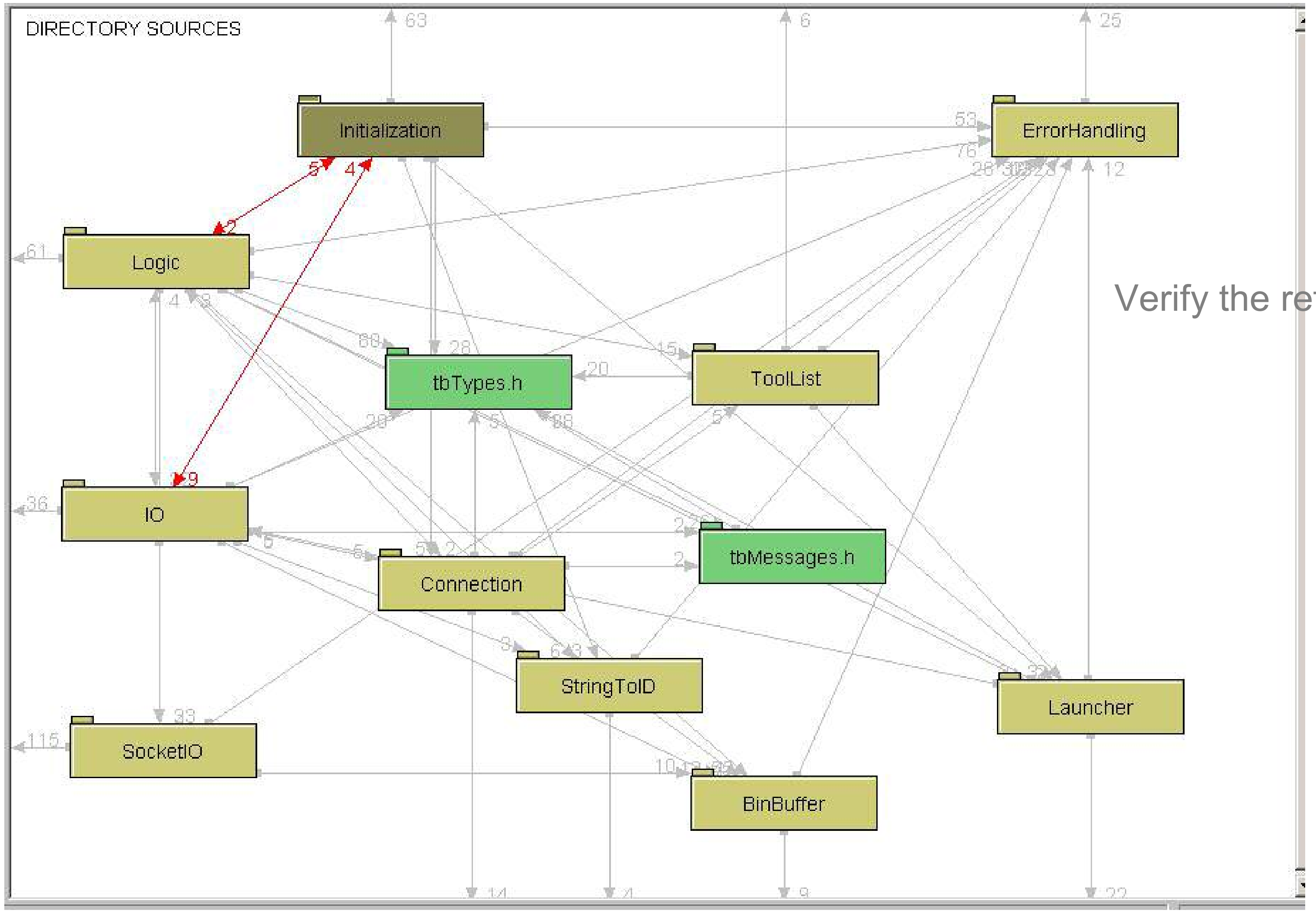
Refactoring (5/6)



Cut misplaced symbols.

Paste the symbols to a new container (file tbLog.C).

Refactoring (6/6)



Verify the refactoring effect.

What next?

- **Architecture model is recovered and evolves alongside with the code**
  - Automatically updated as changes are made to the code
  - Refactored and improved periodically
- **Use the model to accumulate and disseminate knowledge about the software (from architects to developers)**
  - Annotations are added to the model (one-stop-shop)
  - Understand the impact of changes
  - Training of new personnel
- **Proactively enforce integrity of the code using the model**
  - Tools are available (desktop, or integrated into ClearCase)
- **Use the model to harvest reusable components and manage the software product line**
- **Use the model to launch transition to MDA**
- **Plan new feature development using the model**
- **Use the model to plan modernizations**

Remaining challenges

- **Mechanics vs the need to understand the software**
- **Complexity**
  - Use complementary architecture views
  - In order to improve efficiency, focus on use cases
  - Focus on selected components
- **Keep the balance between refactorings for comprehension and “real” refactorings**

Key points: Klocwork Architecture Excavation

- **Focus:**
  - Containers, interfaces, dependencies on top of entities and relationships
- **Container Models**
  - Are scalable and can be used for both abstraction and refactoring
  - Not UML, because of scalability, the need to evolve with the code, and specific “existing code” understanding concerns, like links to the code, navigation, etc.
  - Transition to UML is straightforward
- **Strategy**
  - Top-down
  - As deep as necessary, as shallow as possible
  - Model is built manually by abstracting from code
  - Model is updated and improved incrementally
  - Model is automatically updated as changes are made to the code
- **Steps of the methodology:**
  - Aggregation of entities into bigger containers
  - Refactoring (moving entities between containers)

Key points (cont'd)

- **Model visualizes the effects of architecture erosion**
- **Model allows understanding code**
- **Eliminates erosion:**
  - Restore architecture cohesion, therefore slowing down architecture erosion
  - Proactively slow down erosion by impact analysis
  - Plan modernizations to improve the robustness of the software infrastructure as the new features are being added
- **Model preserves and accumulates the architectural knowledge about the software**
- **Architecture model is shared among the whole development team**
- **Model is changed to drive new development**