How to Harvest Reusable Components in Existing Software

Nikolai Mansurov
Chief Scientist & Architect
Overview

- Introduction
- Reuse, Architecture and MDA
- Option Analysis for Reengineering (OAR)
- Architecture Excavation and Refactoring
- Harvesting as architectural refactoring
  - First, you need to excavate your architecture, as a precise model
  - Harvesting requires the following steps:
    - Identify/localize/collect
    - Isolate component
    - Introduce a boundary
    - Implement refactoring
  - Particular challenges
    - Challenges of component identification (need to understand)
    - Challenges of component isolation
    - Challenges of soft boundaries
    - Reusable components and architecture erosion
    - Reusable components and cyclic dependencies
- Examples
- Conclusions/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
  - Fix bugs
  - Adapt to new operating environments
  - Improve quality

From Ian Sommerville, Software Engineering, 2000
Evolution of existing code has significant economic impact

- 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

From Deursen, Klint, Verhoef, 1999
Software production with existing code

- Changes to existing code may have different *magnitude*:
  - Traditional maintenance (repair)
    - Perfective maintenance
    - Adaptive maintenance
    - Corrective 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
Changes to existing software involve reuse

From Seacord, Plakosh, Lewis, SEI, 2003
The levels of reuse

None

Informal code reuse

Black-box code reuse

Managed workproduct reuse

Architected reuse

Domain-specific reuse-driven Organization (Software Product Line)

Benefits

Investment, experience, time

- Improved time-to-market, costs, quality
- Interoperability, high reuse levels
- Rapid custom development
- Broader coverage
- Reduced maintenance costs
- Reduced development time
- Rapid custom development

Adapted from Jacobson, et.al., 1997

© 2004, Klocwork Inc.
Kinds of reuse

- “As is” reuse
- “Black-box” reuse
- “White box” reuse
- Workproduct reuse
Reuse in multi-release production

From Schach, Tomer, 2001
Product Lines add another dimension to production

From Schach, Tomer, 2001
Internal and External reuse

Reuse within one organization (product line)

Reuse of common components in interoperating systems
- commons computations,
- domain objects
- common protocol modules

© 2004, Klocwork Inc.
Reuse and COTS components

Incrementally created architecture/domain model

modularization

proprietary components

middleware

proprietary components

standardization

COTS component

reuse/buy

© 2004, Klocwork Inc.

Automated Solutions for Understanding and Perfecting Software
Harvesting, standardization, reuse

- (Porting to standard COTS middleware)
- Modularization
- Harvesting of reusable components
- Standardization of components
- Reuse/buy
- Emergent architecture/domain model
Overview

- Reuse, Architecture and MDA
  - Reuse does not involve any translation
  - What can be reused as opposed to what can be translated
    - reuse at low levels and at higher levels
The SEI HorseShoe model

From Smith, et.al., SEI, 1999
Model Driven Architecture

From R. Soley, OMG, 2003

© 2004, Klocwork Inc.
What is Model Driven Architecture?

- A New Way to Specify and Build Systems
  - Based on modeling with UML
  - Supports full lifecycle: analysis, design, implementation, deployment, maintenance, evolution & integration with later systems
  - Builds in Interoperability and Portability
  - Lowers initial cost and maximizes ROI

- Applies directly to the mix you face:
  - Programming language
  - Operating system
  - Network
  - Middleware
Building an MDA Application

Start with a Platform-Independent Model (PIM) representing business functionality and behavior, undistorted by technology details.

A Detailed Model, stating Pre- and Post-Conditions in OCL, and Semantics in Action Language

From R. Soley, OMG, 2003
Generating Platform-Specific Model

Map a PIM to Specific Middleware Technologies via OMG Standard Mappings

MDA tool applies a standard mapping to generate *Platform-Specific Model* (PSM) from the PIM. Code is partially automatic, partially hand-written.

From R. Soley, OMG, 2003
Mapping to Multiple Deployment Technologies

MDA tool applies an standard mapping to generate *Platform-Specific Model* (PSM) from the PIM. Code is partially automatic, partially hand-written.

From R.Soley, OMG, 2003
Generating Implementations

MDA Tool generates all or most of the implementation code for deployment technology selected by the developer.

From R.Soley, OMG, 2003
Integrating Legacy & COTS

Reverse-engineer existing application into a model and redeploy.

MDA Tools for reverse engineering automate discovery of models for re-integration on new platforms.

From R.Soley, OMG, 2003
A typical architecture for Business Reuse

Application systems

Business-specific components

Middleware components

System software components

Adapted from Jacobson, et.al., 1997
Reuse at different layers

- Application logic
- Business objects
- Platform-specific implementation
- Implementation of domain objects
- System services, infrastructure

presentation

application layer

business objects

database
Complete life-cycle: initial release and post-build evolution

new development

value

evolution

evolution challenges

new development challenges

© 2004, Klocwork Inc.

Automated Solutions for Understanding and Perfecting Software
Maintenance in a complete life-cycle

New development

Maintenance

Ability to change
Modernization in a complete life-cycle

- Modernization
- Release 1
- Release 2
- New development
- Maintenance
  ... this includes redevelopment

© 2004, Klocwork Inc.
Managed Architecture in complete life cycle

new development  maintenance  Architecture management

© 2004, Klocwork Inc.
Modernization: Big-bang transformation vs Refactoring

- Transformation
- Maintenance
- New development
- Refactoring

© 2004, Klocwork Inc.
MDA addresses the challenges of a complete life-cycle

...by addressing program comprehension challenge...

...and modernization challenges...

new development

maintenance=continued development
Managed Architecture can kick-start MDA

new development

maintenance
Mega life-cycle: Software Product Lines

SPL evolution challenges

SPL value

single product value

new development

SPL evolution

Release 1

Release 2

Release 3

single product value

SPL value

new development
Software Product Lines are architecture-centric

new development

core assets

SPL evolution

applications

SPL launched

Release 1

architecture
SPL require tight architecture management
Architecture Capability Maturity

- Single aspect of SEI-CMM – refers to the capability of an organization to manage architecture in the complete life-cycle
- **Level 1:** *Initial architecture*
- **Level 2:** *Repeatable architecture*
  - Some packaging rules, some use of libraries, some code reuse
- **Level 3:** *Defined architecture*
  - Components and their interfaces are formally defined and maintained together with the rest of the code,
  - modelling tools like Rational Rose are used
  - middleware or component environment is used,
- **Level 4:** *Managed architecture*
  - Visualization of existing software is available, feedback between the “as designed” architecture and the “as built” architecture is available, metrics of existing architecture are used, architecture integrity is enforced
- **Level 5:** *Optimizing architecture*
  - On-going architecture improvement is part of the overall development process
Overview

- Decision-making process related to reuse

- Option Analysis for Reuse
  - Overview of OAR
  - OAR tasks
  - Example artifacts

- OAR is a systematic, architecture-centric, decision-making method for mining existing components. OAR is used for modernization or redevelopment of existing system.

- OAR is developed in SEI, by Dennis Smith, Liam O’Brien, et.al.
Overview of OAR

From Smith, et.al., SEI, 1999
OAR: Establish Mining context

- **What:**
  - Interview stakeholders
  - Study existing product line or new system requirements
  - Study existing software assets
  - Understand expectations for mining legacy components

- **This establishes**
  - A baseline set of goals
  - Expectations
  - Component needs

- **This uncovers the program and technical drivers for making decisions**

Adapted from Smith, et.al., SEI, 1999
OAR: Inventory Components

What:
- Identify product line needs
- Evaluate legacy components according to screening criteria
- This results in an inventory of candidate components

Tasks:
- Identify characteristics of component needs
- Identify components satisfying criteria
  - Create component table of the legacy components with details of these characteristics
  - Screen components that do not satisfy the required characteristics
- Match components to product line needs
- Update the Inventory with more details on candidate components
- Elicit mining issues and concerns
- Review OAR schedule

Adapted from Smith, et.al., SEI, 1999
OAR: Analyze Candidate Components

- **What:**
  - Analyze the set of legacy components for the types of changes that are required to mine

- **Tasks:**
  - Select desirable components
    - Determine desirability criteria
    - Screen out components that do not satisfy desirability criteria
  - Identify “As Is” and “Black-Box” (wrapped) components
  - Identify “White-box” components (need to be modified)
  - Determine required changes
    - Types of changes each component needs, cost and effort involved, the level of difficulty and risk, and the comparative cost and effort of developing components from scratch
  - Elicit mining options
  - Review OAR schedule

Adapted from Smith, et.al., SEI, 1999
OAR: Plan Mining Options

- **What:**
  - Develop alternatives to mining based on schedule, cost, effort, risk and resource considerations
  - Screen candidate components one more time
  - Analyze the impact of different component aggregations

- **Tasks:**
  - Select favorable components
    - Determine criteria, such as cost or effort
    - Screen out components that do not satisfy criteria
  - Perform component trade-offs
    - Identify one component or combination per product line need
  - Form component options
    - Develop criteria for aggregation
    - Aggregate components
  - Determine comparative cost and effort
  - Analyze difficulty and risk
  - Elicit mining options
  - Update OAR schedule

Adapted from Smith, et.al., SEI, 1999
OAR: Select Mining Option

- **What:**
  - Select the best mining option or combination of options by balancing program and technical considerations
  - Prepare report

- **Tasks:**
  - Choose best option
    - Determine drivers for selecting among options
    - Select an option or a combination of them
  - Verify option
  - Identify component needs satisfied
    - Complete the final list of component needs satisfied and not satisfied through options selected
  - Present findings
  - Produce summary

adapted from Smith, et.al., SEI, 1999
## Example of Component Table

<table>
<thead>
<tr>
<th>Part 1 of 3</th>
<th>Legacy System Software</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPONENT NEEDS</td>
<td>Legacy System Software</td>
<td>Program Language</td>
<td>Directory Name</td>
<td>Number of Modules</td>
<td>Compilable</td>
<td>Size (LOC)</td>
</tr>
<tr>
<td>Sim1 Gateway</td>
<td>C</td>
<td>cag</td>
<td>17</td>
<td>Yes</td>
<td>2,297</td>
<td>Low</td>
</tr>
<tr>
<td>Sim2 Gateway</td>
<td>C</td>
<td>dwg</td>
<td>57</td>
<td>Yes</td>
<td>4,877</td>
<td>Low</td>
</tr>
<tr>
<td>Message Gateways</td>
<td>C-Source</td>
<td>C</td>
<td>csg</td>
<td>45</td>
<td>Yes</td>
<td>4,186</td>
</tr>
<tr>
<td>Truth Model</td>
<td>Objects</td>
<td>Fortran</td>
<td>ojo</td>
<td>65</td>
<td>Yes</td>
<td>9,202</td>
</tr>
<tr>
<td></td>
<td>Event</td>
<td>Fortran</td>
<td>tho</td>
<td>39</td>
<td>Yes</td>
<td>4,412</td>
</tr>
<tr>
<td>ETS &amp; Message Generation</td>
<td>Event Track</td>
<td>C</td>
<td>tsd</td>
<td>89</td>
<td>Yes</td>
<td>5,417</td>
</tr>
<tr>
<td>Man Machine Interface</td>
<td>TBD</td>
<td></td>
<td></td>
<td></td>
<td>30,391</td>
<td></td>
</tr>
<tr>
<td>Executive</td>
<td>TBD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From Smith, et.al., SEI, 1999

© 2004, Klocwork Inc.

Automated Solutions for Understanding and Perfecting Software
## Example of Component Table

**Part 2 of 3**

<table>
<thead>
<tr>
<th>Legacy System Software Components</th>
<th>Characteristics</th>
<th>Characteristics</th>
<th>Characteristics</th>
<th>Characteristics</th>
<th>Characteristics</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-Source</td>
<td>Low</td>
<td>High</td>
<td>BB</td>
<td>None</td>
<td>OK</td>
<td>S&amp;D Files</td>
</tr>
<tr>
<td>Objects</td>
<td>High</td>
<td>High</td>
<td>WB (minor)</td>
<td>Minor (10%)</td>
<td>Adjust</td>
<td>S&amp;D Files</td>
</tr>
<tr>
<td>Event</td>
<td>High</td>
<td>High</td>
<td>WB (minor)</td>
<td>Minor (10%)</td>
<td>Adjust</td>
<td>S&amp;D Files</td>
</tr>
<tr>
<td>Event Track</td>
<td>Low</td>
<td>Low</td>
<td>WB (major)</td>
<td>Major (50%)</td>
<td>Adjust</td>
<td>S&amp;D Files</td>
</tr>
<tr>
<td>TBD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*From Smith, et.al., SEI, 1999*
Example of Component Table

<table>
<thead>
<tr>
<th>Software Components</th>
<th>Legacy System Software</th>
<th>Mining Effort (mm)</th>
<th>Mining Cost</th>
<th>New Development Effort (mm)</th>
<th>New Development Cost</th>
<th>Comparative Cost of Mining</th>
<th>Comparative Effort of Mining</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sim1 Gateway</td>
<td></td>
<td>0.1</td>
<td>$1,000</td>
<td>7.7</td>
<td>$76,567</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>Sim2 Gateway</td>
<td></td>
<td>0.1</td>
<td>$1,000</td>
<td>16.3</td>
<td>$162,567</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>C-Source</td>
<td></td>
<td>0.1</td>
<td>$1,000</td>
<td>14.0</td>
<td>$139,533</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>Objects</td>
<td></td>
<td>2.2</td>
<td>$22,000</td>
<td>30.7</td>
<td>$306,733</td>
<td>7%</td>
<td>7%</td>
</tr>
<tr>
<td>Event</td>
<td></td>
<td>1.3</td>
<td>$13,000</td>
<td>14.7</td>
<td>$147,067</td>
<td>9%</td>
<td>9%</td>
</tr>
<tr>
<td>Event Track</td>
<td></td>
<td>8</td>
<td>$80,000</td>
<td>18.1</td>
<td>$180,567</td>
<td>44%</td>
<td>44%</td>
</tr>
<tr>
<td>TBD</td>
<td></td>
<td>12</td>
<td>$118,000</td>
<td>101</td>
<td>$1,013,033</td>
<td>12%</td>
<td>12%</td>
</tr>
</tbody>
</table>

From Smith, et.al., SEI, 1999

© 2004, Klocwork Inc.

Automated Solutions for Understanding and Perfecting Software
## Example of Options Table

<table>
<thead>
<tr>
<th>Option No.</th>
<th>Legacy System Software Components</th>
<th>Support Software Required</th>
<th>Level of Risk</th>
<th>Level of Difficulty</th>
<th>Mining Effort(^1) (mm)</th>
<th>Mining Cost(^1)</th>
<th>New Development Effort (mm)</th>
<th>New Development Cost</th>
<th>Comparative Cost of Mining</th>
<th>Comparative Effort of Mining</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Event Track</td>
<td>S&amp;D Files</td>
<td>Low</td>
<td>4</td>
<td>8</td>
<td>$80,000</td>
<td>18.1</td>
<td>$180,567</td>
<td>44%</td>
<td>44%</td>
</tr>
<tr>
<td></td>
<td><strong>Option Summation</strong></td>
<td></td>
<td>Low</td>
<td>4</td>
<td>8</td>
<td><strong>$80,000</strong></td>
<td><strong>18.1</strong></td>
<td><strong>$180,567</strong></td>
<td><strong>44%</strong></td>
<td><strong>44%</strong></td>
</tr>
<tr>
<td>2</td>
<td>Objects</td>
<td>S&amp;D Files</td>
<td>High</td>
<td>2</td>
<td>2.2</td>
<td>$22,000</td>
<td>30.7</td>
<td>$306,733</td>
<td>7%</td>
<td>7%</td>
</tr>
<tr>
<td>Event</td>
<td>S&amp;D Files</td>
<td>High</td>
<td></td>
<td>2</td>
<td>1.3</td>
<td>$13,000</td>
<td>14.7</td>
<td>$147,067</td>
<td>9%</td>
<td>9%</td>
</tr>
<tr>
<td></td>
<td><strong>Option Summation</strong></td>
<td></td>
<td>High</td>
<td>2</td>
<td>3.5</td>
<td><strong>$35,000</strong></td>
<td><strong>45.4</strong></td>
<td><strong>$453,800</strong></td>
<td><strong>8%</strong></td>
<td><strong>8%</strong></td>
</tr>
<tr>
<td>3</td>
<td>Sim1 Gateway</td>
<td>S&amp;D Files</td>
<td>Low</td>
<td>1</td>
<td>0.1</td>
<td>$1,000</td>
<td>7.7</td>
<td>$76,557</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>Sim2 Gateway</td>
<td>S&amp;D Files</td>
<td>Low</td>
<td></td>
<td>1</td>
<td>0.1</td>
<td>$1,000</td>
<td>16.3</td>
<td>$162,567</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td>C-Source</td>
<td>S&amp;D Files</td>
<td>Low</td>
<td></td>
<td>1</td>
<td>0.1</td>
<td>$1,000</td>
<td>14</td>
<td>$139,533</td>
<td>1%</td>
<td>1%</td>
</tr>
<tr>
<td></td>
<td><strong>Option Summation</strong></td>
<td></td>
<td>Low</td>
<td>1</td>
<td>0.3</td>
<td><strong>$3,000</strong></td>
<td>38</td>
<td><strong>$378,667</strong></td>
<td><strong>1%</strong></td>
<td><strong>1%</strong></td>
</tr>
</tbody>
</table>

\(^1\) Note: Mining Effort and Cost do not include effort and cost to convert scripts and data files that are part of the support software.

From Smith, et.al., SEI, 1999
Overview
- Architecture Excavation and Refactoring
Excavated high-level architecture
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
Overview

- **Harvesting as architectural refactoring**
  - First, you need to excavate your architecture, as a precise model
  - Reuse requires the following steps:
    - Identify/localize/collection
    - Isolate component
    - Introduce a boundary
    - Implement refactoring
  - Particular challenges
    - Challenges of component identification (need to understand)
    - Challenges of component isolation
    - Challenges of soft boundaries
    - Reusable components and architecture erosion
    - Reusable components and cyclic dependencies
Overview

- Examples of Isolating Components via Refactoring
<table>
<thead>
<tr>
<th>uses identifier</th>
<th>identifier</th>
<th>in File</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept.h</td>
<td>bnnbuf.h</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>bnnbuf.h</td>
<td>bnnbuf.h</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>blog.h</td>
<td>blog.h</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>BinBufferSend</td>
<td>bnnbuf.h</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>inString</td>
<td>bnnbuf.h</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>getLength</td>
<td>bnnbuf.h</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>marnEOB</td>
<td>bnnbuf.h</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>BinBufferRecv</td>
<td>bnnbuf.h</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>askToBuffer</td>
<td>bnnbuf.h</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>getRst</td>
<td>bnnbuf.h</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Error_SocketRecv</td>
<td>bnnbuf.h</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Error_SocketSend</td>
<td>bnnbuf.h</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Error_CreateServerSocket</td>
<td>bnnbuf.h</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>Error_AcceptNewConnection</td>
<td>bnnbuf.h</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Error_ConnectToError</td>
<td>bnnbuf.h</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>ACCEPT3_TYPE</td>
<td>bnnbuf.h</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>eplog</td>
<td>blog.h</td>
<td>24</td>
<td>1</td>
</tr>
<tr>
<td>eplog</td>
<td>blog.h</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>eplog</td>
<td>blog.h</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>socket</td>
<td><em>UNDEFINED</em>@</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>AF_INET</td>
<td><em>UNDEFINED</em>@</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>SOCK_STREAM</td>
<td><em>UNDEFINED</em>@</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>htons</td>
<td><em>UNDEFINED</em>@</td>
<td>12</td>
<td>2</td>
</tr>
</tbody>
</table>
Conclusions/Key points

- Reuse activities add another dimension to multi-release software production
  - Reuse for maintenance (cloning)
  - Reuse for modernization
  - Reuse for redevelopment
  - Reuse for product line (across products)
- Reuse is an architecture-centric activity
- Suitable Architecture Models are required to manage reuse
- Harvesting is an architecture refactoring
  - First, you need to excavate your architecture, as a precise model
  - Harvesting requires the following steps:
    - Identify/localize/collect
    - Isolate component
    - Introduce a boundary
    - Implement refactoring