Reuse of Open Source Software

Ioannis Stamelos

Aristotle University of Thessaloniki
School of Informatics
Software & Interactive Technologies Lab

stamelos@csd.auth.gr

Presentation Outline

- **Introduction - Open Source and Reuse**
- Systematic Reuse & Open Source Components
- Recent Research Developments – ICSR Conference
- Software recycling
- Conclusion and Future work

Current Situation in Open Source

- Hundreds of thousands of open source software applications, estimated 600.000+:
  - Sourceforge alone: >300.000 projects, more on GitHub
  - 100+ billion lines of code
- Written in many different languages, owned by communities, open to everyone
- Open source is ~50% of deployed code
- (source: 2012 Future of Open Source Survey, Black Duck Software)

Software Issues

- Software (either open or closed source) is often pestered by a multitude of problems:
  - Bad quality (bugs, low usability, unclear behaviour, ...)
  - Bad (internal) documentation (purely commented code, no analysis&design documentation, no recorded tests ...)
  - In addition: Original developers may be no longer available, many OSS projects are not active at all

Software Reuse

- “Code reuse, also called software reuse, is the use of existing software, or software knowledge, to build new software” (Frakes, Kang, 2005)
- Oppportunistic vs planned reuse. In planned reuse “a team strategically designs components so that they'll be reusable in future projects” (wikipedia)
- In closed source systematic reuse mainly occurs in Software Product Lines (SPLs)

Reuse in Closed Source Sofware

- In Closed Source, reuse at the class level has been critisized: expensive, little impact, not easy (classes/objects are too detailed to be reused). Higher design levels are considered more appropriate for reuse.
- Components are cohesive sets of classes that provide and require interfaces to offer specific services to client software applications
  - An example may be a Data Collector to be used in sensor network, an Address Finder, a Library Manager, etc.
- Reuse at a component level has been proposed and Component Based Software Engineering has emerged as a SE discipline.
- Design pattern solutions may also be seen as reuse mechanisms.

Open Source Reuse

- "Open-source software offers the most astounding range of reusable assets for any software project", Alan W. Brown and Grady Booch (2002)
- Code is extensively reused in open source, e.g:
  - Apache Commons
  - The FFmpeg project that has produced several components reused in >140 other OSS projects (Capiluppi, Stol, Boldyreff, IJOSSP, 2011)
  - ROS, Robot Operating Systems tools and libraries
- Considering the volume of open source code, reuse occurs mainly in an opportunistic and non systematic way
- Planned Reuse in Open Source, is it possible?

Presentation Outline

- Introduction - Open Source and Reuse
- **Systematic Reuse & Open Source Components**
- Recent Research Developments – ICSR Conference
- Software recycling
- Conclusion and Future work

Component characteristics in closed source (1)

- Standardised: a component has to conform to some standardised component model
  - This model may define component interfaces, component meta-data, documentation, composition and deployment
- Independent: it should be possible to compose and deploy it without having to use other specific components.
- Composable: all external interactions must take place through publicly defined interfaces
  - In addition, it must provide external access to information about itself such as its methods and attributes

Component characteristics in closed source (2)

- Deployable: a component has to be self-contained and must be able to operate as a stand-alone entity on somecomponent platform that implements the component model.
  - This usually means that the component is a binary component that does not have to be compiled before it is deployed.
- Documented: Components have to be fully documented so that potential users of the component can decide whether or not they meet their needs
  - The syntax and, ideally, the semantics of all component interfaces have to be specified

Component Issues in Closed Source

- Trust
  - An untrusted component may not operate as advertised; at worst, it can become a security breach
  - Requirements may not fit those of the client software applications
- Validation
  - The component specification may not be detailed enough to allow comprehensive tests to be developed
  - Components may have unwanted functionality

Component Issues in Open Source (1)

- Suppose that a critical mass of Open Source components become available. The aforementioned issues become easier to address.
- Trust: becomes not an issue, as
  - Source code may be inspected and it may be verified that it poses no security or other problems
  - In case requirements do not fit, the reuser may opt to modify the component, producing a new component version
- White box tests may be designed and run to validate the component
- Unwanted functionality may be spotted and if necessary removed, producing a new component version

Component Issues in Open Source (2)

However other issues may arise

- Companies are often the developers and maintainers of component models (e.g. Microsoft and .NET)
  - If component models need to be used, Open Source Communities should provide them
  - Domain architectures may be a solution: a community defines a domain architecture on which open source components may be easily integrated
- Documenation is often neglected in open source. A non documented component may be useless.
- Patent issues, what happens to the client application that reuses open source components?

Presentation Outline

- Introduction - Open Source and Reuse
- Systematic Reuse & Open Source Components
- **Recent Research Developments – ICSR Conference**
- Software recycling
- Conclusion and Future work

Recent Developments in Reuse Research

- International Conference on Software Reuse – ICSR
- Some interesting research results have emerged in the recent editions of ICSR
  - ICSR 2015 – Miami, US
  - ICSR 2016 – Cyprus, CY
  - ICSR 2017 – Salvador, BR
  - ICSR 2018 – Madrid, ES

ICSR 2015 (also on a JSS special issue)

- **Automating the license compatibility process in open source software with SPDX** (Best Paper)
  - Georgia Kapitsaki, Frederik Kramer and Nikolaos Tselikas use SPDX files in open source to resolve license compatibility issues and detect license violations
  - In addition, they propose valid combinations of open source packages and licenses
- **Construction and Utilization of Problem-Solving Knowledge in Open Source Software Environments**
  - Hyung-Min Koo and In-Young Ko build a Bayesian network for constructing a knowledge base in a semi-automatic way
  - Knowledge comes in terms of causes, symptoms and solutions to problems related to open source code and its reuse

ICSR 2016

- **A Case Study On The Availability Of Open-Source Components For Game Development**
  - Maria-Eleni Paschali, Apostolos Ampatzoglou, Stamatia Bibi, Alexander Chatzigeorgiou, Ioannis Stamelos
  - 110 OSS games were analyzed for offering reuse opportunities in the computer gaming domain. Reusability was measured through QMOOD.
  - Games with complex logic (FPS, strategy, sports, role playing) were found to be more appropriate for reuse
  - The most reusable assets were characters and scenarios

ICSR 2018

- **Reusability Index: A Measure for Assessing Software Assets Reusability** (Best Paper)
  - Apostolos Ampatzoglou, Stamatia Bibi, Alexander Chatzigeorgiou, Paris Avgeriou and Ioannis Stamelos
  - A composite metric that synthesizes seven reusability aspects (structural quality, external quality, documentation, etc.)
  - Results on OSS projects show that RI performs better than previously proposed simpler metrics

More OSS Reuse Research (1)

- **Impact of Internal Open Source Development on Reuse: Participatory Reuse in Action**, Vitharana, King, Chapman, Management Information Systems, 2014
  - Internal Open Source in IBM fosters the development of reusable assets
- **Using Code Skeleton Patterns for Open Source Reuse**, in Advances in Computer Science and Ubiquitous Computing, Springer, 2017
  - Nam, Kim, Hong propose the use of skeleton patterns to "... improve reusability by reducing the modification of open source software."

More OSS Reuse R&D (2)

- Journal publications from AUTH PhD Students (Kakarontzas, Ampatzoglou, Constantinou, Kritikos)
  - An empirical investigation on the reusability of design patterns and software packages
  - Building and mining a repository of design pattern instances
  - Extracting reusable components: A semi-automated approach for complex structures
  - Reusability of open source software across domains
  - Open source reuse processes
- Tools:
  - PARSEWeb: A Programmer Assistant for Reusing Open Source Code on the Web, Thummalapenta Xie, Automated Software Engineering, 2007
  - Code search engines: Koders, Merobase, Google Code Search, etc.
  - EU/OPENSME project

Presentation Outline

- Introduction - Open Source and Reuse
- Systematic Reuse & Open Source Components
- Recent Research Developments – ICSR Conference
- **Software recycling**
- Conclusion and Future work

Recycling

- **Recycling** is processing used materials (waste) into new products to prevent waste of potentially useful materials, reduce the consumption of fresh raw materials, reduce energy usage, ... [source:wikipedia]
- To **reuse** is to use an item again after is been used
- Recycling is not always economically efficient or even environmentally helpful

Software Recycling

- **Software Recycling** is processing **existing** (either used or dormant) software to help producing **new software products**, to prevent waste of potentially useful **development effort** and reduce **software development costs**

Software Recycling vs Software Reuse

- **Software Reuse** is the process of creating software systems from predefined **software components**
- **Software Recycling** is the process of systematically building reusable software components out of existing software, even if the original application was not meant or designed to be reused. Such components may then be reused into new software systems

What to Recycle?

- Open Source Code
  - Need to develop a recycling mindset among communities
- Legacy Systems
  - Only legacy system owners have such option
  - However, they may want to donate legacy code under certain circumstances
  - An example could be the public sector: hundreds of software applications that may not be used and could be released as open source (e.g. recent decision by the State of California to embrace open source)
- In any case one must carefully consider the software quality issues mentioned above
  - to make software recycling economically efficient and
  - to produce components of acceptable quality

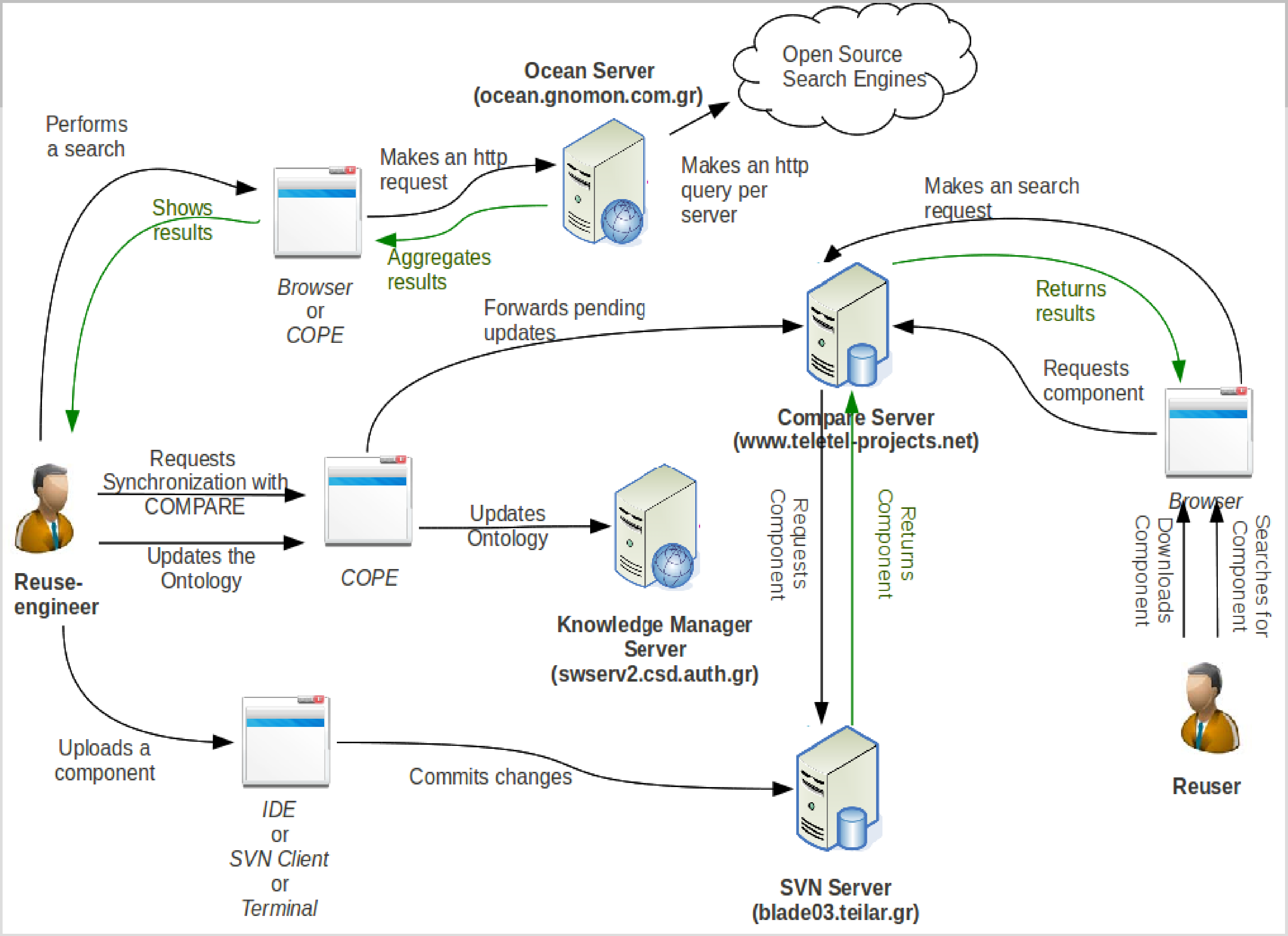
Who could/should recycle?

- Source: COMPARCH 2012/ROSS workshop, Bertinoro (IT), June 25
- Large companies :
  - Would like to recycle their own code (e.g. IBM, SIEMENS)
  - Reluctant to use recycled open source components or third party software because of liability issues ("not invented here" syndrom)
- SMEs have less liability issues because they address broad markets and/or produce less critical applications
  - Eager to reuse recycled software
  - Practical problems exist...

Technical Challenges

- Valuable OSS components may exist in every OSS project.
- However it is difficult to recognize them, extract them, test them, document them etc.
- During software development, usually there is no time for the aforementioned activities. Developers often prefer to develop new code from scratch although this code has been written before many times by many others.
- Even when developers recognize the opportunity to reuse OSS code there are several uncertainties related to the provided functionality and quality.
  - What the component does?
  - How well it does it?

EU/OPENSME Project Overall Architecture (2010-2012)



Software recycling business models

- Several business models are possible:
  - Recycling SME associations offer recycling service for free to member SMEs, charging other SMEs or large companies
  - Recyling service offered for a subscription fee
  - Repository access is charged
  - Pure open source models are also possible, based on additional services, like component maintenance, testing, documentation, training etc.
- Pricing models must be established

Total Process Cost (TPC)

TPC =

$$\sum_{i=1}^{i}\left[\left(b_i * Lc_i\right) + A_i\right] + \left(\frac{1-R}{N}\right) * \sum_{j=1}^{j}\left[\left(b_j * Lc_j\right) + A_j\right]$$

May be used for pricing calculations, e.g. how much a reuse client should be charged for the reuse of a specific component

Total Cost of Process elements

TCP = Total Cost of Process

b = Relative cost of engineering activities - represents the fraction of cost per activity. (b is is an estimation and can be provided through simulations (Monte-Carlo) or by regression analysis.)

Lc = Labor cost for Reuse Engineer

A = Activities additional overheads

R = Proportion of adaptation activities requested ($R \leq 1$)

N = Number of reuses over which the asset development costs will be amortized

i = the range of additional requested activities (e.g. certification)

j = the range of all activities minus the i activities

Presentation Outline

- Introduction - Open Source and Reuse
- Systematic Reuse & Open Source Components
- Recent Research Developments – ICSR Conference
- Software recycling
- **Conclusion and Future work**

Concluding remarks

- Active, high quality, well documented OSS projects may provide reusable assets
- A large portion of open source is completely inactive and adds to the confusion and incertainty around OSS.
- Components may be extracted from existing OSS projects (even if inactive), recycling code and knowledge. More domain specific components means more value from reuse.
- Research on reuse may provide useful tools for developing, locating, reusing, maintaining open source components
- Business models are needed to support the above and encourage developers to dedicate effort for OSS component development

Future Work

- Merge reuse technology advancements and open source development
  - Build search engines, tools, platforms to facilitate OSS reuse
  - Further support the reuse engineer cognitive process for isolating components (e.g. through semantic analysis on the code, program comprehension techniques)
  - Investigate approaches for builiding reusable OSS components at the first place
- Identify application domains where reuse has better chances to be successful
- Business modeling for OSS reuse
- Recycling cost modeling
- Financial/dynamical modeling of the open source software recycling process
- Explore OSS for teaching reuse practices to CS students
- Increase interaction between the reuse and open source communities (e.g. by co-organizing OSS and ICSR or making a working group?)

THANK YOU FOR ATTENDING

QUESTIONS