Code Harvesting with Zeligsoft CX

Zeligsoft

November 2008

zeligsoft

Code Harvesting with Zeligsoft CX

Code harvesting with component modeling increases software reuse and improves developer efficiency for embedded systems applications. Zeligsoft's Component Oriented Engineering (COE) methodology, CX development tool suite and expertise can be used to harvest legacy application code and transform it into a component-based, model-driven digital asset. This can reduce code size by 20-40% and development efforts by 40-60%.

1 INTRODUCTION

Embedded Systems development teams are under ever increasing pressure to reduce their application development time and project costs while continually improving both the quality and reliability of their software-intensive products. Conventional approaches to achieving this are to increase software reuse and improve software developer efficiency by employing software abstraction and automation.

Software reuse provides benefits such as:

1. Reduced design and coding effort due to a decreased requirement for new hand crafted software;
2. Improved quality and reduced debugging effort resulting from the re-application of proven software components; and
3. Increased proportion of investment (effort) in new value added application software.

It has been estimated[1] that reusing software requires on average 20% of the effort of new development. Development teams can apply modeling and code generation tools to further increase the benefits of reuse by:

1. Reducing the effort required to reuse code to 5% or less;
2. Enabling greater reuse of existing code (i.e. increase the amount of software that can be reused for 5-20% of the cost of new code); and
3. Increasing the reusability of new code (i.e. improve the potential for future reuses).

Code harvesting legacy software is one way to increase the amount and effectiveness of reuse in development projects. Existing, proven code is a valuable asset that can be exploited. Zeligsoft's methodology and modeling tools enable the effective harvesting of these assets — maximizing reuse and increasing developer efficiency.

2 ZELIGSOFT COE AND CX

Component-Oriented Engineering (COE) is the software development methodology designed by Zeligsoft to address the challenges of complex embedded systems characterized as distributed applications that target heterogeneous multiprocessor platforms. COE is based on the best practises of component-based development (CBD) and Model Driven Development (MDD)[2].

The COE approach involves developing and creating individual components of code which are assembled in a model to construct a higher-level function, e.g. an application. The formal separation of software elements (components), along with the clear specification of possible communication relationships (ports and connections), enables reliable construction of applications and improved software reuse. COE allows for abstraction of the application and platform layers of the system, affording developers greater flexibility and possibilities for reuse across other software applications. When combined with the automatic

generation of portions of the structure, control, and platform specific software, COE allows developers to concentrate their efforts on the value-added functional (behavioural) code of the application.

Zeligsoft CX is a powerful, ready to use "out of the box" toolkit that comes bundled with embedded system domain specializations and integration with the IBM/Rational RSM tool. CX offers development teams openness and user control via its standards-based architecture and customizable domain-specific modeling and languages.

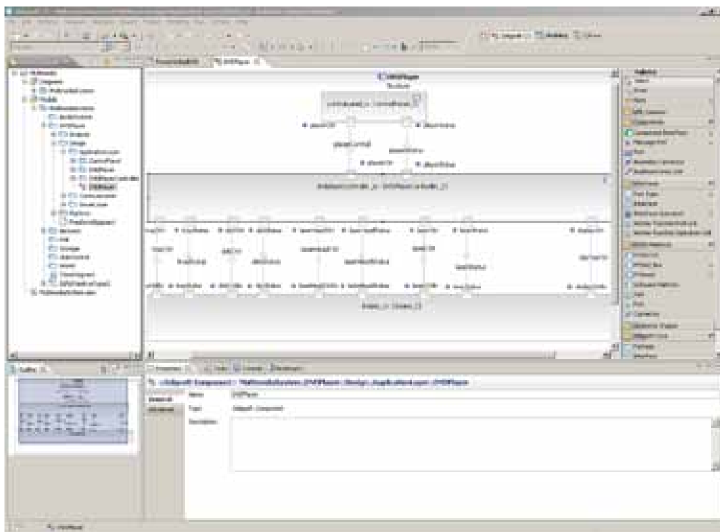

Figure 1: Zeligsoft CX

3 CODE HARVESTING WITH COE/CX

Harvesting a legacy application with Zeligsoft's COE methodology and CX tool suite produces:

- A model of the application that reflects the actual design and implementation;
- A repository of the application's components;
- Abstractions (profiles) of the logical and physical platforms which are the basis for application deployments;
- Component infrastructure code that is generated for each deployment based on a standard Zeligsoft design pattern; and
- Code generated for other customer-specific or domain-specific design patterns relevant to the application.

Harvesting existing software using COE/CX offers the opportunity for significant code reuse and efficiency improvements from:

1. Reduction of the equivalent amount of application code — replacing code by:
    a. Reusing existing components.
    b. Generating code for recurring design patterns.

2. Future project savings:
    a. Reduced software lifecycle costs of the smaller application code base and the use of model-driven development with advanced tooling.
    b. Recurring productivity improvements such as tool support for re-partitioning or platform re-targeting.

3. Other project savings from the reuse of harvested components across product lines.

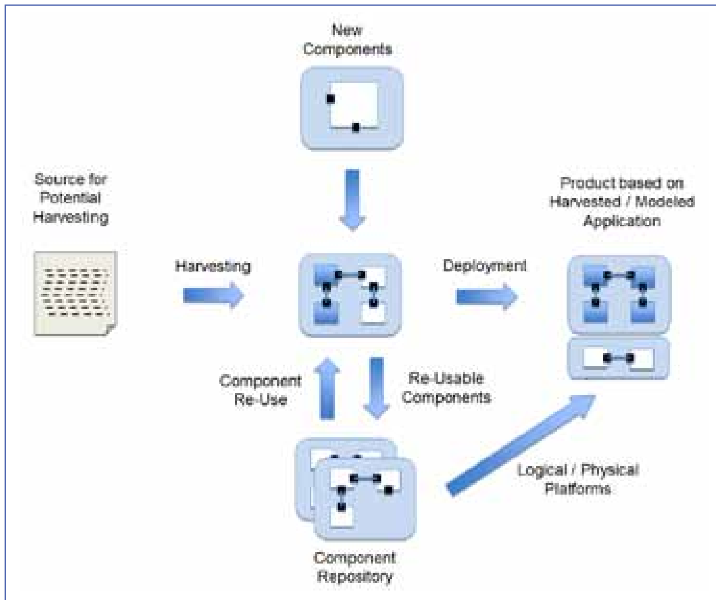

Figure 2: Code Harvesting Process

The effectiveness of a given harvesting project will depend on a number of factors:

Is the existing application component-based?
The architecture of a component-based or object-oriented architecture can remain unchanged. A legacy application that is not component-based requires extra consideration as to how the architecture will be modelled, as there may be restrictions imposed by the legacy code.

Is there an existing repository of components?
If so, these components could be used to replace equivalent functionality in the legacy application — reducing the resulting SLOC (Source Lines of Code) of the harvested application.

Have applications or platforms already been created/harvested using COE and Zeligsoft tools?
This will significantly reduce the effort required to reuse existing designs and code in the legacy application as:

- Existing components will already be modelled and packaged for reuse;
- A component infrastructure will be in place;
- Platform abstractions and realizations will be readily reusable or extended; and
- Implementations of recurring design patterns will be readily available to the designers.

By component modeling an application's design and code using CX, the reuse effort will be effectively "drag and drop".

4 QUANTIFYING THE SAVINGS FROM CODE HARVESTING WITH COE/CX

It is clear that harvesting existing designs using COE and CX, and bringing them into the CX environment, can provide the many qualitative benefits described above.

Quantifying the benefits of harvesting from what is in effect re-writing an existing, working application is required to prove the value of a harvesting project and the Zeligsoft approach.

A simple but effective method for deriving this net benefit is to compare the cost, in terms of ongoing development effort, of the existing (legacy) application to that of the harvested application.

Product software has a cost associated with it. Development effort (measured by developer person months) is needed to develop, enhance, and sustain software throughout its lifecycle. The amount of development effort is a function of the code size, the type of development project, and the efficiency of the development team.

The project costs of the two scenarios (legacy application versus harvested application) do not need to include the costs of full system testing on their actual embedded platform(s). This is because they are assumed to be equivalent, regardless of code base, since the product functionality is the same.

This quantification compares the design, coding, integration, and testing costs of the two (2) application software code bases (legacy versus harvested).

4.1 SLOC

Source Lines of Code (SLOC) is a simple metric for code size. A benefit of a code harvesting project is the resulting reduction in the equivalent SLOC of the harvested application compared to the original (legacy) application.

By reducing the "SLOC load" on a development team, developer time savings will be accrued upon completion of the harvesting as well as over the lifetime of the application software. Any reuse of components (in other projects) derived from the harvested application code creates savings attributable to the harvesting.

4.2 Person-months per KLOC

A standard metric for developer productivity is person-months per KLOC (1,000 SLOC). For embedded systems, this can vary from 1 to 5 person-months per KLOC for new application development. The number (1-5) for a given project (and team) depends on the complexity of the software and the tooling available to the developers.

Transforming a legacy application into a COE/CX application will improve the efficiency of the development team by reducing the person-months required per KLOC. This is because once the application has been harvested into CX, the benefits of model-driven development and domain specialization apply.

4.3 Types of development activities

All lines of code do not require equal development effort. Activities relating to maintaining code, for example, take much less effort than new application development. To effectively factor in the difference in activity type as it relates to development effort, a weight (effort as compared to new application development) is used against the code. The different types of development activities and their relative efforts are:

- New application (100%)
- Enhancement (40 to 60% of new application development)
- Re-development (20 to 40% of new application development)
- Maintenance (5 to 10% of new application development)

For example, a code base of 1,000 source lines of code with an average developer productivity of 3 person-months per KLOC, where the development activity is primarily enhancements, equates to 1 x 3 x 50% = 1.5 person months.

5 MODEL FOR ESTIMATING CODE HARVESTING SAVINGS

Given a quantity of legacy application code (measured in SLOC), we can estimate the savings in development effort (person-months) resulting from harvesting the code into CX by calculating the difference in cost of the legacy code versus the harvested code.

The cost of each code base is measured by multiplying the code size (in KLOC) by the effort (person-months per KLOC) and factoring in the required weight for the particular development activity. See sections 4.1, 4.2 and 4.3 above.

The application code is also categorized to account for the efforts and the benefits of harvesting different types of code. The proportions of each category for a given application **(% of Appl.)** are used to weight these efforts and benefits.

For this model the following categories are used:

Reused — the proportion of the original application that can be replaced with existing components;

Generated — the code that can be replaced by tool-generated artifacts;

Application — the application-specific (unique) portion of the code that will remain **(= 100% - Re-used – Generated)**

Deployment — the percentage of the code related to deployment of the application; and

Reusable — the proportion of the application code that could potentially be reused in other projects.

Savings will occur based on:

Release — reducing the effort required for a given project type. This will almost entirely be effort required by the post-harvesting application code;

Deployment — eliminating code and effort that would need to be changed for every product partitioning or platform. This code would either be reused (from a previous project) or generated in accordance with a pattern; and

Reuse — as future reused code in other projects and applications.

The benefits of harvesting with COE/CX are estimated by calculating the:

1. **Original-Code** size for the category of code **(= Application x % of Appl)**. For example: if there are 100-KLOC of original application code of which 20% can be generated then:

   $\text{Original-Code}_{\text{Generated}} = \text{100-KLOC} \times 20\% = \text{20-KLOC}$

2. **Original-Cost (= Original-Code x Effort)**.
   For example: if enhancement of code costs a team (on average) 1.5 months per KLOC and there are 20-KLOC of code then the cost without the benefits of harvesting would be:

   $\text{Original-Cost}_{\text{Generated}} = \text{20-KLOC} \times 1.5 = \text{30 person-months}$

3. **Harvested-Code** equivalent to the original **(= Original-Code x % of Category)**. For example: if there are 100-KLOC of original code, 20% of which can be generated by the CX tool, and 10% will remain after harvesting (won't be artifact code), then:

   $\text{Harvested-Code}_{\text{Generated}} = \text{20-KLOC} \times 10\% = \text{2-KLOC}$

4. **Harvested-Cost** factoring in the reduction of code and effort **(= Harvested-Code x Original-Effort x % of Effort)**. For example: if enhancement of code costs a team (on average) 1.5 months per KLOC and COE/CX reduces the effort to 10% of the manual approach:

   $\text{Harvested-Cost}_{\text{Generated}} = \text{2-KLOC} \times 1.5 \times 10\% = \text{0.3 months}$

5. The delta from the **Original-Cost** is the estimated **Saving**. For the generated code in this example, the savings would be:

   $\text{Original-Cost}_{\text{Generated}} - \text{Harvested-Cost}_{\text{Generated}} = 30 – 0.3 = \text{29.7 person-months}$

Note: the example estimated saving of over 95% of the cost reflects the difference between manually enhancing 20,000 SLOC of code versus applying modern tooling to 2,000 SLOC of application-specific (value-added) code. While the system and product testing efforts would be the same, there are considerable savings to be had from code reuse and generation during the development phase.

6 SAMPLE SAVINGS CALCULATIONS

The following are the calculations for a simple sequence of representative development scenarios:

1. The initial harvesting of a legacy subsystem and application;
2. Harvesting a complete application combined with code enhancement (e.g. new features) and some intra-project reuse; and
3. Developing new application software using COE/CX with an existing collection of components available for reuse.

For these examples, we have used the following parameters for the model. These values are based on industry averages. They could be adjusted to better suit a specific project, application and development team.

Type of Development Project	Weight (% of New Code Development)	Effort p-m / KLOC
New Code	100.0%	3.000
Enhancement	50.0%	1.500
Re-development	30.0%	0.900
Maintenance	7.5%	0.225

Table 1: Weighting of Project Types and Development Efforts

Code Type	% of Code after Harvesting	% of Effort due to COE/CX
Re-used	20%	25%
Generated	5%	10%
Application	100%	75%
Deployment	10%	20%
Reusable	20%	25%

Table 2: Impact of Code Harvesting and COE/CX Approach

6.1 Design Harvesting — Legacy Application

Harvesting of existing application in a green field scenario:

Legacy Application - Green Field	Original Code / Approach				Harvested Code — COE/CX				Saving
	% of Appl.	Code	Effort	Cost	% of Category	Code	% of Effort	Cost	person-months
100,000		KLOC	p-m / KLOC	person-months		KLOC		person-months	
per Release									
Re-used	0%	0.0	0.225	0.0	20%	0.0	25%	0.0	0.0
Generated	25%	25.0	0.225	5.6	5%	1.3	10%	0.0	5.6
Application	75%	75.0	0.225	16.9	100%	75.0	75%	12.7	4.2
		100.0		22.5		76.3		12.7	9.8
per Deployment									
Deployment	15%	15.0	0.900	13.5	10%	1.5	20%	0.3	13.2
per Reuse									
Reusable	10%	10.0	0.900	9.0	20%	2.0	25%	0.5	8.6

Notes:
- This is a green field scenario, therefore there are no components to reuse
- Assumed to be in maintenance mode (a conservative assumption)

Analysis:
- The cost of the harvested application is almost recovered if a small subset of the resulting components is reused
- The harvesting cost for subsequent application code will benefit from economies of scale — the component repository will be built up
- Additional deployments could be created for optimization purposes (this would be prohibitively expensive without COE/CX)

6.2 Application Re-development

The harvesting scenario for application code undertaken at the same time as an enhancement project:

Legacy Application Enhancement	Original Code / Approach				Harvested Code — COE/CX				Saving
	% of Appl.	Code	Effort	Cost	% of Category	Code	% of Effort	Cost	person-months
100,000		KLOC	p-m / KLOC	person-months		KLOC		person-months	
per Release									
Re-used	10%	10.0	1.500	15.0	20%	2.0	25%	0.8	14.3
Generated	25%	25.0	1.500	37.5	5%	1.3	10%	0.2	37.3
Application	65%	65.0	1.500	97.5	100%	65.0	75%	73.1	24.4
		100.0		150.0		68.3		74.1	75.9
per Deployment									
Deployment	15%	15.0	0.900	13.5	10%	1.5	20%	0.3	13.2
per Reuse									
Reusable	10%	10.0	0.900	9.0	20%	2.0	25%	0.5	8.6

Notes:
- This is a green field scenario but there is assumed to be component reuse within the context of the project
- The effort is increased to account for the enhancement activities (as opposed to code maintenance)

Analysis:
- The CX tool's deployment-aware code generation provides the greatest benefit
- Almost all the effort is spent (focused) on the application-specific code
- Even a modest amount of reuse of components resulting from harvesting can provide significant savings to other development projects

6.3 New Application Development

Scenario for developing a new application using COE/CX and drawing on an existing component repository:

Legacy Application - Green Field	Original Code / Approach				Harvested Code — COE/CX				Saving
	% of Appl.	Code	Effort	Cost	% of Category	Code	% of Effort	Cost	person-months
100,000		KLOC	p-m / KLOC	person-months		KLOC		person-months	
per Release									
Re-used	20%	20.0	3.000	60.0	20%	4.0	25%	3.0	57.0
Generated	25%	25.0	3.000	75.0	5%	1.3	10%	0.4	74.6
Application	55%	55.0	3.000	165.0	100%	55.0	75%	123.8	41.3
		100.0		300.0		60.3		127.1	172.9
per Deployment									
Deployment	10%	10.0	0.900	9.0	10%	1.0	20%	0.2	8.8
per Reuse									
Reusable	10%	10.0	0.900	9.0	20%	2.0	25%	0.5	8.6

Notes:
- This could be a new standalone application or an extension to the previous application
- A below average amount or future reuse is assumed
- Given the deployment flexibility offered by COE/CX, it is assumed that the application would exploit this capability and a larger portion of the functionality would be deployment related

Analysis:
- The development effort is almost entirely focused on the 55% of unique (value added) functionality

7 CONCLUSIONS

1. Harvesting provides immediate and quantifiable benefits — code size can be reduced by 20-40%;
2. Zeligsoft's COE methodology and CX tooling can increase the benefits of harvesting — reducing development efforts by 40-60%, providing deployment flexibility, and enabling future reuse; and
3. The benefits of harvesting with COE/CX compound over time — savings occur with every release, deployment, and reuse.

8 CUSTOM ANALYSIS

Interested parties are encouraged to contact Zeligsoft to obtain a copy of this model so that they can use their own metrics to quantify the potential savings from using Zeligsoft COE and CX to harvest their application code.

Glossary

COE	Component-Oriented Engineering
COTS	Commercial Off-the-Shelf
IP	Intellectual Property
KLOC	Thousand (1,000) Source Lines of Code
RTOS	Real-Time Operating System
SLOC	Source Lines of Code

7 REFERENCES

1. *Measuring Software Reuse: Principles, Practices, and Economic Models* by Jeffery Poulin. ISBN-13: 978-0201634136
2. *Component-Oriented Engineering...the dawn of a new era in embedded software development productivity* by Francis Bordeleau and Ross MacLeod, Zeligsoft Inc.
3. *Zeligsoft CX Product Description*

zeligsoft

Contact Information

Website: www.zeligsoft.com

Email: info@zeligsoft.com

Toll-free (North America): 1-800-ZELIGSW (1-800-935-4479)

Direct dial: +1 819-684-9639

Zeligsoft, Zeligsoft CE and Waveform Builder are trademarks of Zeligsoft Inc.
All other trademarks are the property of their respective companies.

Version 1.1

Printed in Canada © Copyright 2005.