Application of SADT for source code generation in learning the programming fundamentals

Kustov M.[1], Guban B., Datsun N.[2]

Department of Applied Mathematics and Informatics
Donetsk National Technical University
Donetsk, Ukraine
[1]maksym.kustov@gmail.com, [2]datsun@pmi.dgtu.donetsk.ua

Abstract— **Approach to generating source code from the SADT (Structured Analysis and Design Technique) specification of the program is offered. Invariants are allocated in basic algorithms. Algorithm of generating source code on the basis of templates is formulated. Data structures used for implementation of algorithm are considered. Internal representation of the SADT specification can be used to analyze the properties of the generated program.**

programming fundamentals, SADT, algorithms, invariants, data structures, generating source code

I. INTRODUCTION

This paper presents the results of application of the SADT (Structured Analysis and Design Technique) [1, 2] specification for the generating source code. The course SE101 of SEEK (Software Engineering Education Knowledge) [3] is focused on the solution of such learning objectives: develop simple statements of requirements and write small programs in some language. Modern methods of generating source code from a Domain-Specific Language (DSL)/specification language (SL) can be used in learning to create a structured program based on the standard algorithms and simple data structures. Development of methods for generating source code from a DSL/SL is an actual problem for the various subject areas [4-10]. There are textual and visual DSL/SL. UML and UML-like languages are most popular among the visual specification languages. C++, C#, Java, VB and VB.NET in most cases are the source code languages [4-9]. But the notation of UML and object-oriented languages for generating source code are redundant for problems of learning the programming fundamentals. SADT [1, 2] as a methodology for functional modeling successfully works for the description of clearly specified processes [11, 12]. Therefore it is possible to apply SADT in learning the methods of decomposition programs. Language SADT-methodology is a visual specification language. Graphical notation of IDEF0 [2] in such subject area as formalization of programs development allows to consider the logic relations between functional blocks of the program and provides the minimum toolkit for building software projects with standard algorithms. Work objective is to use the graphic notation of SADT for teaching of decomposition program and generating source code for procedural language from the SADT-specification with standard algorithms on basic data structures.

II. SELECTION OF INVARIANTS OF BASIC ALGORITHMS

A. Selection of Invariants of The Algorithm With Respect To The Input Data

Let's try to identify the invariants for basic search algorithm of the maximum negative elements. For this we consider the implementation of this algorithm for various data structures. Fig. 1 presents these algorithms for the array, a text file and linked list.

f=0;	**f=0;**	**f=0;**
i=0;	fseek(file, 0L, SEEK_SET);	i=p;
	fscanf("%d",&a);	// p - head list
while(i<n)	while(!feof(file))	while(i!=NULL)
{	{	{
a=x[i];		a=i->info;
if((a<0)&&!f))	**if(a<0&&!f)**	**if((a<0)&&!f)**
{	**{**	**{**
f=1;	**f=1;**	**f=1;**
res=a;	**res=a;**	**res=a;**
}	**}**	**}**
if(f&&(a<0) && (res<a))	**if(f&&(a<0) &&(res<a))**	**if(f&&(a<0) &&(res<a))**
res=a;	**res=a;**	**res=a;**
i++;		i=i->next;
}	}	}
if(!f)	***if(!f)***	***if(!f)***
{	***{***	***{***
printf("There are no right elements");	***printf("There are no right elements ");***	***printf("There are no right elements ");***
}	**}**	**}**
Algorithm A1	Algorithm B1	Algorithm C1

Figure 1. Basic algorithms of the maximum negative element for the array, a text file and linked list.

After detailed analysis of algorithms, you may notice that some parts are the same for all types of data structures. Let's distinguish immutable parts (invariants) of algorithms.

In Fig. 1, the invariants of these algorithms are marked in bold. Thus we have the invariants of algorithms with respect to the input data. That is, when you change structure of input data invariant remains the same. The resulting invariants are divided into three parts: initialization (in Fig. 1 marked in underlined), main part, ending part (in Fig. 1 marked in italic).

Unchanging part of the processing algorithm determines the input variable and the variable part of it represents this variable to the input, looking over an array, file or linked list.

B. Selection of the algorithm invariant by the method of processing

We consider algorithms for finding maximum of the negative elements, minimum and replacement of the positive elements of zeros for the same data structure (array). Fig. 2 presents these algorithms.

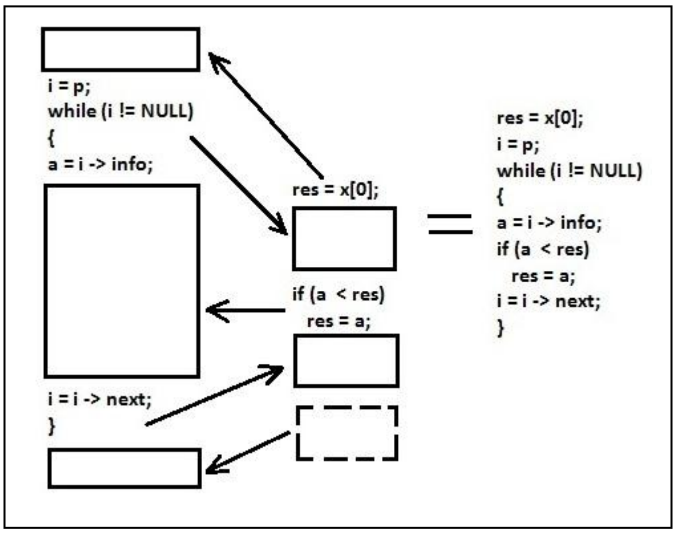
f=0;	res=x[0];	
i=0;	**i=0;**	**i=0;**
while(i<n)	**while(i<n)**	**while(i<n)**
{	**{**	**{**
a=x[i];	**a=x[i];**	**a=x[i];**
if((a<0)&&!f)		
{		
f=1;		
res=a;		
}		
if(f&&(a<0)&&	if(a<rez)	if(a>0)
(res<a))		
res=a;	res=a;	x[i]=0;
i++;	**i++;**	**i++;**
}	**}**	**}**
if(!f)		
{		
printf("There are		
no right elements");		
}		
Algorithm A2	Algorithm B2	Algorithm C2

Figure 2. Various algorithms for the array.

Let's distinguish the invariants for the considered algorithms. Common to these algorithms is the cycle for the elements of the array. In Fig. 2, the invariants of these algorithms are marked in bold. In these algorithms there are no some parts: algorithm B2 has lost the ending part, and the algorithm C2 lost the initialization and the ending parts.

III. Generation of New Algorithms Based on Invariants

Let's create a patterns of the algorithms obtained in the previous sections. To do this, we will cut out the invariants from the algorithm. Now we can get a brand new algorithm by combining the algorithm C1 and algorithm B2. Fig. 3 illustrates the preparation of an entirely new algorithm based on invariants. Further, the project implementation software for generating program code based on the specifications of the SADT-methodology will be discussed.



```
i = p;
while (i != NULL)
{
a = i -> info;
```

```
i = i -> next;
}
```

```
res = x[0];
```

```
if (a  < res)
  res = a;
```

```
res = x[0];
i = p;
while (i != NULL)
{
a = i -> info;
if (a  < res)
  res = a;
i = i -> next;
}
```

Figure 3. Process of obtaining a new algorithm based on invariants.

IV. Data Structures

From the IDEF0/SADT graphic language for creation of the SADT specification are used:

- functional blocks (Activity Box) to denote the program modules at decomposition
- interface arcs (Arrow) of two types (“Input” to transfer the input data and “Output” to transfer the output data);
- names of interface arcs (Arrow Label) to denote the names of the input and output data.

Consider the representation of the visual components of SADT specification in the form of data structures

A. SADTUnit Structure

The main data structures is SADTUnit, describing a unit of SADT diagrams, and SADTData, describing the type and data structure of links SADT diagrams. Elements of structure SADTUnit are:

- number of diagram unit (integer);
- links (list);
- containers (list).

“Number of diagram unit” serves for process ordering. Process occurs consistently, according to numbers of diagrams.

List item "Links" is a pair <key (name); value (link structure)>. This component will organize a logical relationship of component with other diagram units.

List item "Containers" is a pair <key (name); value (container structure)>. It contains a set of possible patterns of code for generated program. Needed container is selected in accordance with the data type and structure of the input data.

B. *SADTLink Structure*

Elements of structure SADTLink are:

- name (string);
- type of link (string);
- linked to the diagram (string);
- link on the external diagram (string);
- data (string).

"Name" is used to display in graphical representation of the diagram. Also the name is used for this purpose what to refer to this communication from external diagrams and communication with various internal structures, such as the mask of using of the diagram container.

"Type of link" specifies the type of communication with respect to the diagram in which this connection is contained. Link can be:

- input (intended to link a diagram to input data);
- output (intended to create of the reference to the data received as a result of work of the diagram);
- modular (intended to connect the various modules to the diagram, which can extend standard functionality).

"Linked to the diagram" specifies the name of an external diagram. It is used in input and modular links to specify the diagram connected with the current diagram through this link.

"Link on the external diagram" is the name of the output link of external diagram, which is connected with this link. It is used for input and modular links.

The field "Data" is used for output links. The value of this field corresponds to a name of data which are created as a result of work of the current unit diagram. This field establishes compliance between data and links.

If the diagram generates new data, the fields "Linked to the diagram" and "Link on the external diagram" are empty, and the field "Data" refers to the structure of the newly created data. Such a set is created only for the output links. Conversely, if the field "Data" is empty, and the fields "Linked to the diagram" and "Link on the external diagram" are filled, it means that the link is connected with one of the external diagrams, or that the link is associated with one of the inputs to the current diagram. This occurs when new data aren't generated, but only only to modify the input data.

C. *SADTData Structure*

Elements of structure SADTData are:

- data structure (string);
- data type (string);
- name of the data (string).

"Data structure" describes the type of the data structure (array, list, file).

"Data type" describes the type of data in structure (integer, character or real).

"Name of the data" used to identify the created data structure.

By analogy to SADT diagrams, SADTUnit itself acts as a diagram, and SADTData acts as links between diagrams. If we consider the mechanism of processing of the workpiece in the SADT specification, SADTData will act as a workpiece, and SADTUnit will act as methods and techniques for handling the workpiece. Accordingly, after processing of the workpiece is obtained by the final product, which will be the output link. It will be at the same time presented by SADTData structure.

D. *Container Structure*

Elements of container structure are:

- text field (string);
- mask of using (array of structures).

"Text field" contains one of versions of the generated text of the program.

Structure of the field "Mask of using" contain the name, structure and type of input links. The container contains some such structures. If all parameters (structure and type), specified in the mask, correspond to all the listed links, this particular container is used for source code generation.

E. *Mask of Using Structure*

The mask of using is the array, each element of which is the data structure:

- name of link (string);
- data type (string);
- data structure (string).

“Name of link” should correspond to one of names of links of diagram units.

Type in the "Data Type" should match the type of the corresponding input link. Then the container will be called, and the code will be generated by its pattern.

"Data Structure" contains the name of the data structure relevant to the input link.

Compliance of a mask of the input link is defined by the coincidence of the input link name of diagram and the name of link in the mask of using.

F. *Data Structures Communication*

The main block is «SADTUnit». It contains the array of structures «Container» and «SADTLink». In turn, the structure of the container contains the array of of structures «Mask of Using», and the field «Data» of structure «SADTLink» refers to a structure «SADTData».

V. Source code generation

A. Description of Problem

Input data:

ST: set /* set of basic data structures */
ST={array, matrix, list, sequential file, direct access file}
AT: set /* set of basic algorithms */
BD: set /* set of diagram blocks */
BD = <nbdid, BDIN, BDOUT, MET, INST>
nbdid: string /* ID of the diagram block */
BDIN: set /* set of inputs of diagram */
BDOUT: set /* set of outputs of diagram */
BDMET: set /* set of diagram methods*/
INST: set /* set of tools */

Restrictions: AT, ST, BD sets are not empty.

Results:

D: graph /* the problem specification from which the file PR is generated */
PR: file /* file of program code */

Relation " input data - results ":

D=BD union V
$V = \{v_i\}$
$v_i = < bdin^k_i, bdout^l_j >$, i, j — numbers of input and output block diagram, k, l - numbers of inputs and outputs
PR= AT × ST × P

B. Source Code Generation by SADT Specification

Using of templates is a common practice in programming. Many programming languages have functions to work with templates. But the usual patterns become powerless when the variables substituted into the templates have different structure (arrays, lists, files), because the processing of each data structures is individual. For the array, it is possible to use the loop, the matrix is required nested loops, for a tree - bypassing the depth or breadth.

Several containers of a template used to solve this problem. Appropriate container is selected according to the type and structure of the input data. After that contents of the container are used as a usual template.

Code generation is a substitution of data from the input links in the text of container. The universality of this approach is that the containers are defined for each data type and data structure.

C. Source Code Generation Algorithm

1. Creation SADT diagrams.

2. Creation of links for SADT diagrams and giving them names.

3. Numbering units in order to code generate.

4. Creation of containers with templates.

5. Specifying of mask of using for each of containers. The algorithms contained in the containers can be created on the basis of invariants of the algorithms.

6. Linkage of the created blocks. Linking of blocks occurs by comparison to the input link of one diagram and the output links of another diagram. Fields «Linked to the diagram» and «Link on the external diagram» are filled. Link will be recursively traced to the reference to data.

7. Selection of the container, which corresponds to the input data. For this purpose it is necessary to compare the data of each input link. Then, knowing type and data structure with which should operate the diagram block, select the appropriate template. From the list of all possible patterns is chosen the one with the mask of using corresponds to the input data.

8. Replacement in a template of the container of values of input links on names of data which correspond to this link.

9. Repeating steps 1-8 to all diagrams.

10. Arrangement of the obtained text of a program code according to numbers of diagrams.

VI. Description of Test Stand

A. Implementation Toolkit

Qt library [13] was chosen as an instrument of implementation. It includes all the basic classes that may be required for the development of applied software. Qt is fully object-oriented tool, easily expanded and supporting of component programming technique. Thus, the designed software product is cross-platform and can easily extended to work over the network and databases.

Development of software is based on the Linux operating system CentOS 5.5. Tests were carried out for the following operating systems CentOS5.5, Fedora 14, Windows XP. Compilation and debugging implemented in QT Creator.

B. Architecture Variants

Experiments were carried out on different hardware architectures running different operating systems. Table I presents options for the test stand.

TABLE I. Variants of Test Stand

Hardware	OS
Intel(R) Celeron(R) CPU E3300 2.5GHz 1GB RAM	Microsoft Windows XP Professional Service Pack 3
AMD Phenom(tm) II X4 B45 Processor 3,1 GHz 1370112 KB RAM	CentOS 5.5 x86_64
AMD Sempron(tm) 140 Processor 2,7GHz 1796440 KB RAM	Fedora 14 x86_64

VII. Testing

A. Description of Test Set

The experiment was performed on data structures: the array of integers, the direct access file of integers, the array of strings.

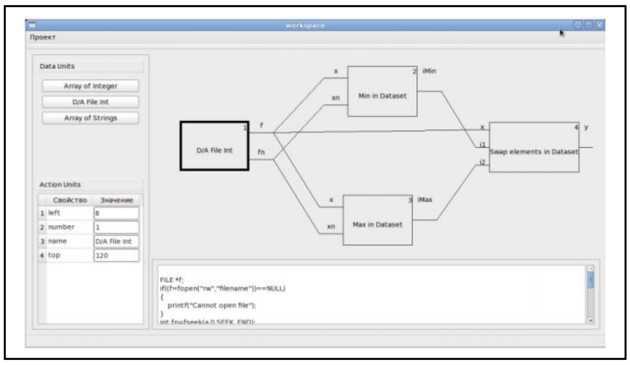
The blocks of diagram with containers ("find the minimum", "find the maximum", "swap elements") were created for processing of these data structures.

Containers are contained in each of blocks of diagrams. Each of the container corresponds to a certain set of input data. Table II presents the set of containers for each of the block.

TABLE II. SETS CONTAINERS FOR EACH BLOCK

Data	Container
	Array of Integer
Один контейнер, создающий данные	int an=20; int a[20];
	Direct Access File of Integers (D/A File Int)
Один контейнер, создающий данные	FILE *f; if((f=fopen("rw","filename"))==NULL) printf("Cannot open file"); int fn=fseek(a,0,SEEK_END);
	Array of Strings
Один контейнер, создающий данные	int cn=20; char c[20][100];
	Minimum in Dataset (Min in Dataset)
ArrayInt	int iMin=0; for (int i=1; i<%xn%;i++) { if(%x%[i]<%x%[iMin]) iMin=i; }
DAFileInt	int iMin=0; int Min,bufMin; fseek(%x%,0,SEEK_SET); fread (&Min, sizeof(Min), 1, %x%); for (int i=1; i<%xn%;i++) { fread (&bufMin, sizeof(Min), 1, %x%); if(bufMin<Min) { Min=bufMin; iMin=i; } }
ArrayString	int iMin=0; for (int i=1; i<%xn%;i++) { if(strcmp(%x%[i],%x%[iMin])&lt;0) iMin=i; }
	Maximum in Dataset (Max in Dataset)
ArrayInt, DAFileInt, ArrayString	By analogy to the container Min in Dataset
	Swap elements in Dataset
ArrayInt	int buf; buf=%x%[%i1%]; %x%[%i1%]=%x%[%i2%]; %x%[%i2%]=buf;
DAFileInt	int changeMax,changeMin; fseek(%x%,iMin*sizeof(int),SEEK_SET); fread (&changeMin, sizeof(Max), 1, %x%); fseek(%x%,iMax*sizeof(int),SEEK_SET); fread (&changeMax, sizeof(Max), 1, %x%); fseek(%x%,iMin*sizeof(int),SEEK_SET); fwrite (&changeMax, sizeof(Max), 1, %x%); fseek(%x%,iMax*sizeof(int),SEEK_SET); fwrite (&changeMin, sizeof(Max), 1, %x%);
ArrayString	char buf[100]; strcpy(buf,%x%[%i1%]); strcpy(%x%[%i1%],%x%[%i2%]); strcpy(%x%[%i2%],buf);

Fig. 4 shows the interface of the test stand.



	Свойство	Значение
1	left	8
2	number	1
3	name	D/A File Int
4	top	120

```
FILE *f;
if((f=fopen("rw","filename"))==NULL)
{
   printf("Cannot open file");
}
int fn=fseek(a,0,SEEK_END);
```

Figure 4. Interface of the test stand.

VIII. EXPANSION OF SET OF BASIC ALGORITHMS

Let's consider source code generation by the SADT specification on an example of search of an element "a" an and further removal of the found element from the one-dimensional array and the list. In Fig. 5, the invariants of these algorithms are marked in bold.

f=0;	**f=0;**
res=-1;	**res=-1;**
i=0;	i=p;
while ((i<n) && !f)	while ((i!=NULL) && !f)
{	{
r=x[i];	r=i->info;
if (r==a) {	**if (r==a) {**
f=1;	**f=1;**
res=i; }	**res=i; }**
else	**else**
i++;	i=i->next;
}	}
Algorithm A3	Algorithm B3

Figure 5. Element search (array and list).

Having analyzed algorithms of A3-B3 and A4-B4, we see that some parts of these algorithms are identical. Fig. 6 shows that the same algorithm is specific to the different data structures.

i=0;	i=p; // p - head list
while (i<Res)	while (i!=Res)
i++;	i=i->next;
while (i<n)	while (i!=NULL)
{	{
a[i]=a[i+1];	i->info=i->next->info;
i++;	i=i->next;
}	}
Algorithm A4	Algorithm B4

Figure 6. Remove with a shift (array and list).

Fig. 7 represents the result of combining these two basic algorithms. Under the proposed approach in this work, it is possible to allocate the invariants of algorithms and to create containers for source code generation for SADT diagrams for the other data structures (direct access files and sequential files, strings or trees).

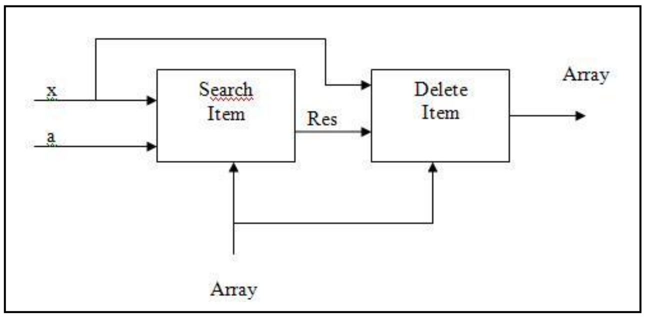


Figure 7. Combining the basic algorithm A3 and A4.

In Fig. 8, the invariants of these algorithms are marked in bold.

f=0;	**f=0;**
Res=-1;	**Res=-1;**
	fseek(file, 0L, SEEK_SET);
i=0;	i=ftell(file);
while(i<strlen(str) && !f)	while ((fscanf(filein, "%d", &r) != EOF) && !f)
{	{
r= str[i];	i=ftell(file);
if (r==a)	**if (r==a)**
{	**{**
f=1;	**f=1;**
Res=i;	**Res= i;**
}	**}**
else	**else**
i++;	i++;
}	}
Algorithm A5	Algorithm B5

Figure 8. Element search (string and file).

The project results will be used for teaching the bachelors "Software Engineering" in the discipline of "Programming Fundamentals."

Prospects for the development of the project involve the solution of such problems:

- analyze generated programs to determine their efficiency;
- add an interface arc "Control" to describe conditions that are imposed on standard algorithms.

IX. CONCLUSION

In learning the programming fundamentals of "Software Engineering" students is necessary to develop their competence of structuring problems. When studying standard algorithms and basic data structures it is important to show the general approaches and implementation features. Therefore, this project focuses on the using of SADT-methodology for learning the programming fundamentals. Possibilities of the specification are limited to standard algorithms and basic data structures. Approach of source code generation by SADT specification is offered. Invariants are allocated in basic algorithms. Source code generation algorithm on the basis of templates is formulated. Possibility of creation of new algorithm on the basis algorithms is provided. Data structures used for implementation of algorithm are considered. Internal representation of the SADT specification can be used to analyze the properties of the generated program.

REFERENCES

[1] D. Ross, "Structured Analysis (SA): A Language for Communicating Ideas", IEEE Transactions on Software Engineering, vol. SE-3, N. 1, pp. 16-34. 1, Jan. 1977.

[2] D.A. Marca and C.L. McGowan, "IDEF0 and SADT: a modeler's guide,", Auburndale, OpenProcess, Inc., 2006, p.392.

[3] "Recommendations for teaching software engineering and computer science in universities = Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering; Computing Curricula 2001: Computer Science," V.L. Pavlov, A.A. Terekhov, and A.N. Terekhov, Eds. Moscow: INTUIT.RU "Online University of Information Technologies", 2007, p. 462 [Рекомендации по преподаванию программной инженерии и информатики в университетах, М.: ИНТУИТ.РУ "Интернет-Университет Информационных Технологий", 2007, 462с.].

[4] S. Kelly, J.-P.Tolvanen, "Domain-Specific Modeling: Enabling Full Code Generation," Wiley-IEEE Computer Society Press, 2008, p.448.

[5] J.-P. Tolvanen, "Domain-specific modeling for full code generation," Journal of Software technology, vol. 12, N. 4, Jan. 2010.

[6] B. Jager and M. Rosenau, "Method for generating source code in a procedural, re-entrant-compatible programming language using a spreadsheet representation,". US patent application 11/057,430, issue date 9/6/2011, patent number 8015481.

[7] N. M. Jakubiak and M. Kucharski, "System and method for generating source code-based test cases," application number 11/558241, publication date 08/16/2011.

[8] J. M. Festa,"Systems and methods for generating source code for workflow platform," patent application number 20100281462, publication date 11/04/2010.

[9] M. Fowler, "Code Generation for Dummies," Methods & Tools, Spring 2009, vol. 17, N. 1, pp 65-82.

[10] K. Vogel, "A source code generator for C: a language-independent means of building programs that are consistent, elegant, and fast," Journal Dr. Dobb's, vol. 16, pp. 28 -35, Aug. 1991.

[11] H.S. Delugach, "A Multiple-Viewed Approach to Software Requirements," Ph.D. Dissertation, Department of Computer Science, University of Virginia, Charlottesville, VA, May, 1991.

[12] O. Djebbi, "Eliciting Requirements Variability for Embedded Real-Time System Family," in Proceedings of the First International Workshop on Situational Requirements Engineering Processes: Methods, Techniques, and Tools to Support Situation-Specific Requirements Engineering Processes (SREP'05), Paris, France, August 29-30, 2005, In Conjunction with the Thirteenth IEEE Requirements Engineering Conference (RE'05), J. Ralyté, P. J. Egerfalk, and N. Kraiem, Eds. Ireland: University of Limerick, 2005, pp. 192-199.

[13] M. Summerfield, "Advanced Qt Programming: Creating Great Software with C++ and Qt 4," Prentice Hall, 2010, p.550.