kleiner kommunikation
max@kleiner.com

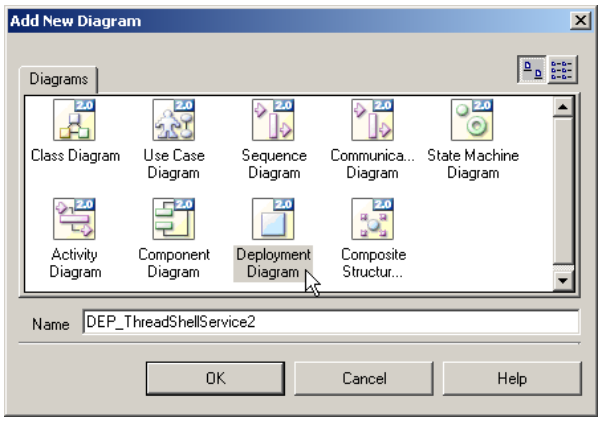
1 Model View in Delphi 2007

Das in Delphi Enterprise (Architect) diskret integrierte Tool von Together, lässt sich bereits nach Minuten einsetzen. Der Model View ist ein Werkzeug, welches gemäß Metamodell und Metriken den Code in UML darstellt, aber auch prüft, analysiert und dokumentiert, bspw. ungenutzte Variablen und private Felder bemängelt oder Namenskonventionen überprüft. Die so generierte tabellarische Auswertung dient der Code-Optimierung und eignet sich für ein Systemhandbuch oder sonstige Review Reports. Viel Expertenwissen bei einfacher Bedienung.

1.1 An der Quelle

Der Begriff Model - View kann auch Teil des MVC Pattern sein, hat aber in diesem Report keine Bedeutung. Hier geht es um den Together Support. Für uns ist die Model View in Delphi eine Einheit, die nebst den Diagrammen auch Audits&Metrics und Patterns beinhaltet. Hier will man wissen, wo sich komplexe Routinen befinden, ob man Namenskonventionen einhält oder wie hoch die Kopplung einer Klasse ist, also auf zum Model View.

Mittlerweile sind alle UML Diagramme enthalten (Klassendiagram und Charts, Sequenzdiagram, Kollaborationsdiagram (seit UML 2 als Composite Structure Diagram), Zustandsdiagram, Verteilungsdiagram, Anwendungsfall (UseCase), Aktivitätsdiagram und Komponentendiagram. Das Paketdiagram ist explizit im Model View als Unit dargestellt und hat auch beim Refactoring einen hohen Stellenwert. Was viele nicht wissen, wie Abb. 2 zeigt, aus einem generierten Paketdiagramm kann ich direkt und übersichtlich, z.B. Klassen in ein anderes Paket (Unit) verschieben und dies mit zugehöriger Code Generierung!



Wer sich vor allem mit UML beschäftigt, so genannte Design-Projekte, dem sei das Verzeichnis `...\Demos\Modeling\UML` empfohlen. Hier wurde ein durchgängiges Projekt modelliert.

Auch in ECO ist ja die Visualisierung schon fast ein Muss. Das in der Architect Version enthaltene ECO 3 ermöglicht es, Klassen mit Together-Technologie zu entwickeln und diese dann für einen Permanentspeicher wie etwa SQL Server, Oracle, Interbase, DB2 oder sogar eine XML-Datei zu verwenden.

Together bietet generell die Möglichkeit, Klassen in UML zu modellieren und diese Modelle dank der LiveCode-Technik unmittelbar in Code umzusetzen. Der eigentliche Trick an LiveCode ist dabei, dass es hinsichtlich des Klassendiagrams keine Rolle spielt, ob man Änderungen nun grafisch am Modell vornimmt oder manuell im Code der zugehörigen Unit / Klassendefinition. LiveCode sorgt dafür, dass sich beide Darstellungen des Modells synchron verhalten. Jede beliebige Klassenhierarchie in Delphi (VCL, FCL, VCL.NET) lässt sich auch auf Basis des Codes per Mausklick visualisieren und automatisch dokumentieren. Die generierten Reports sind auch mit rechter Maustaste aus der Model View erreichbar. Together vereint was man bisher vom Pascal Analyzer (PAL) und ModelMaker kennt;

wobei die beiden Tools immer noch die volle Daseinsberechtigung haben, da der Funktionsumfang der beiden bei weitem noch nicht erreicht ist.

Die Reports in tabellarischer Ansicht (Abb. 2), die man meistens in einem Durchgang erzeugt, enthalten Kennzahlen zur Unterstützung und Bewertung des Codes und seiner Struktur beim Entwickeln des Systems. Einige schauen wir weiter unten an. Damit wollen Sie objektive und vergleichbare Erkenntnisse zu jeder untersuchten Software produzieren, und zwar unabhängig der Sprache. So hält man die Codeinspektion oder ein Review für die effektivste Maßnahme, um die Codequalität verbessern zu können. Ein Teil dieses Wissens steckt in Together.

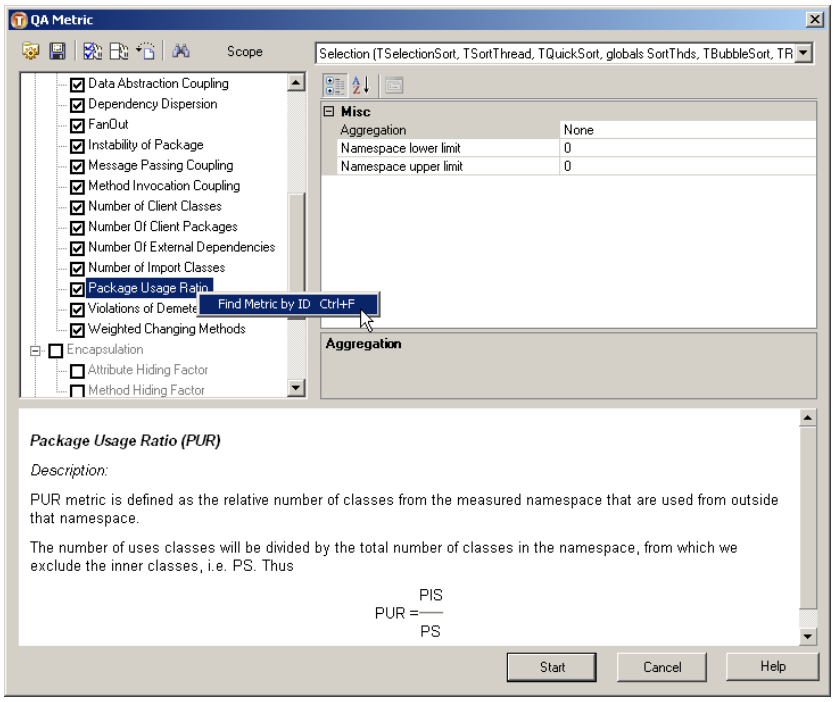


Abb. 1: Interaktive Hilfe im Model View

Together nimmt keine Änderungen am Code vor sowie man selbst keine Modifikationen am Code vornehmen muß. Es sei denn man lässt bewusst Code produzieren, wie Pattern- oder Refactoring Techniken es tun. Am häufigsten kommt Together bei Code Reviews oder als Vorbereitung zur Integration vor, da es das Tuning und Management des Entwicklungsprozesses erleichtert. So läßt sich beim Ändern einer Subroutine schnell mal die davon abhängigen Prozeduren in einer Metrik aufzeigen (z.B. Dependency Dispersion (DD) oder Coupling Between Objects (CBO)).

Im Paket ist auch ein „Find Metrics /Audit by ID“ enthalten (CTRL F), siehe Abb. 1, der sich parametrisieren läßt und eigene Sets erlaubt. Dieser Zusatz ist mächtig, da interaktive Erklärungen auf einen Blick verfügbar sind.

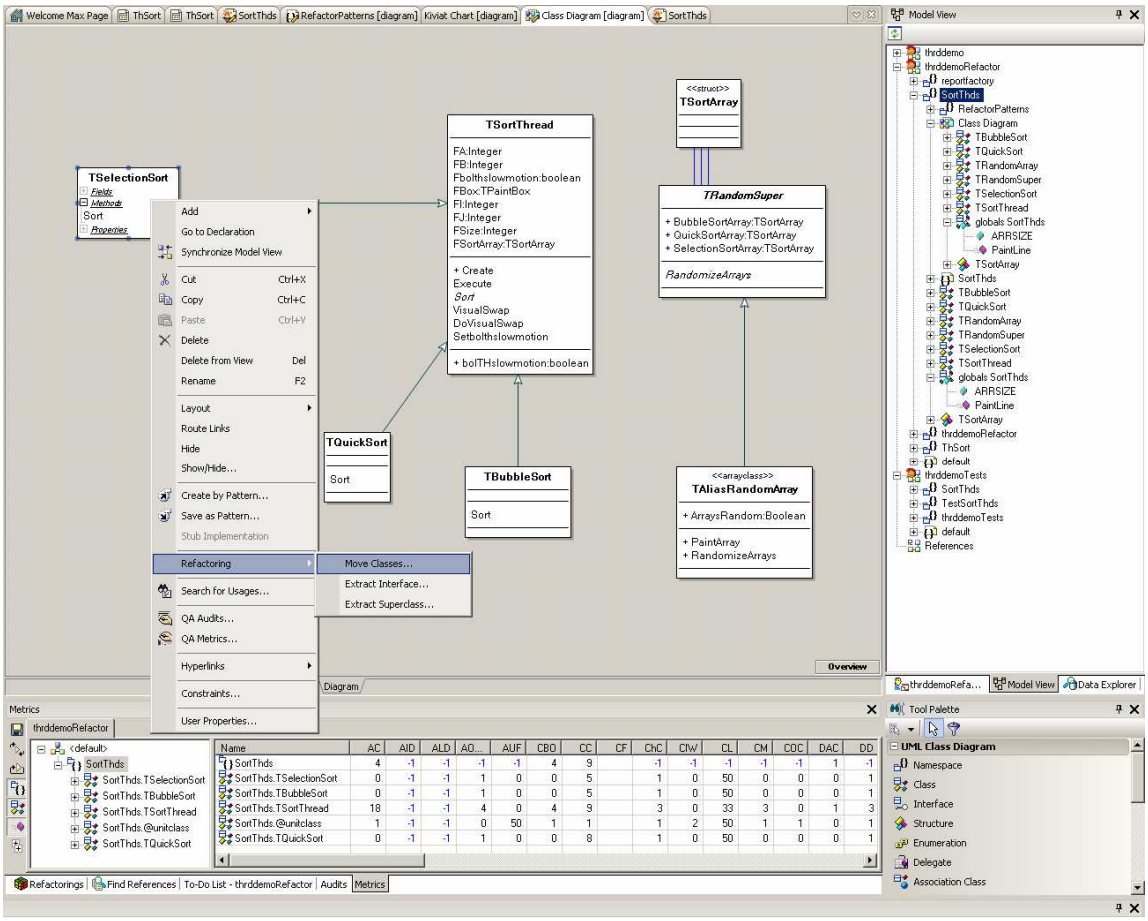


Abb. 2: Metriken im Einsatz

Kommen wir zu einigen Reports und deren Anwendung [1]:

Der **Data Abstraction Coupling** (DAC) zählt die Referenztypen innerhalb einer Klasse und macht Aussagen über die Kopplung zu anderen Klassen. In Abb. 2 ist der DAC auf 1 gesetzt, will heissen in der Klasse `TSortThread` ist die Referenz `TPaintBox` gemeint.

Eine äusserst interessante Metrik (NOprnd) kommt von Halstead, der das Verhältnis von Operatoren und Operanden durchleuchtet:

```
procedure Class1.x(v: boolean);
01 var i,r: Integer;
02    b: Exception;
03 begin
04   if (v) then
05     i:= 1
06   else
07     i:= 2;
08
09   case i of
10     1:
11     2:
12     else
13   end;
14   try
15     while (v) do begin
16       v:= false;
17       r:= 1;
18       b:= Exception.Create('!!!');
19       i:= ((i * i) + r);
20       break
```

[1] Im Folgenden das Delphi Demo Projekt Thread

```
21     end
22   except on e: Exception do
23     raise e
24   end
25 end;
```

Line	N1	n1	N2	n2
04	if	if	v	v
05	=	=	i, 1	i, 1
07	=		i, 2	2
09	switch	switch	i	
15	loop	loop	v	
16	=		v, false	false
17	=		1	
18	=, new, call	new, call	Exception, "!!!"	Exception, "!!!"
19	=, *, +	*, +	i, i, i, r	r
23	throw	throw	e	e
Total	14	9	17	9

- N1 is NOprtr (Number of Operators),
- n1 is NUOprtr (Number of Unique Operators),
- N2 is NOprnd (Number of Operands),
- n2 is NUOprnd (Number of Unique Operands)

In den Anfängen von Together und Delphi 2005 war noch zuviel Fehlinterpretation durch Java vorhanden, weil Together ursprünglich für Java getrimmt wurde. Einige Abweichungen zu ObjectPascal gibt's immer noch:

Beim „Member is Not Used" (MNU) wird in der Prozedur QuickSort die Sub Prozedur als nicht im Gebrauch bemängelt, da Java offensichtlich keine lokalen Prozeduren kennt. Oder einige Felder lassen sich gemäss einem Audit `private` deklarieren, gemeint ist aber `strict private`.

Weiter bei den Warnungen sind Variablen aufgelistet, die wohl referenziert aber nie gesetzt / initialisiert werden oder umgekehrt. Nicht immer aber hat Together Recht, z.B. muß man ein Objekt ohne Felder innerhalb einer Klasse auch nicht allozieren.

Die Gruppe **Design Flaws Coding Style** oder **Expressions Report** bei den Audits hilft Codezeilen zu reduzieren indem man Bezeichner die nie referenziert sind eigentlich entfernen oder aus kommentieren kann. Auch Funktionen / Prozeduren die nur einmal aufgerufen werden sind nicht unbedingt effizient und lassen sich durch Together aufdecken. Bei Subprogrammen, die den Rückgabewert einer Funktion nicht auswerten, legt das Tool ebenfalls ein Veto ein. Wirkungsvoll ist auch die Möglichkeit, alle Codepassagen aufzuzeigen, die eine Variable zweimal setzen obwohl zwischen den Zeilen keine weitere Auswertung derselben erfolgt:

```
procedure MaxBox;
var i: integer;
begin
i:= 7;
..
i:= 10; // !! i is set again
..
if i = 10 then begin
..
```

Wie Steve McConnell 1993 schon in *Code complete* schrieb, sind Kontrollen und Tests eine Weise der Entschädigung der vorweggenommenen menschlichen Fehlbarkeit.

Richtig komplex wird es nun mit dem **Complexitiy Report**, (CC) welcher in der Rubrik Metriken zu finden ist (Abb. 2). Hier kommt die Decision Point Analyse zum Zug die identisch mit der McCabe's Metrik ist, auch cyclometric complexity genannt. Die Berechnung ist transparent dokumentiert und jederzeit auch manuell nachvollziehbar. Nun, was ist der Nutzen? Diese in der Industrie anerkannte Metrik macht eine Aussage über die „Komplexität" und Verständlichkeit des Codes, welcher dann teilweise im Widerspruch zur Wiederverwendbarkeit steht. Bei Werten >10 ist es angebracht, das Subprogramm in kleinere Teile zu zerlegen. Konkret in unserem Thread Beispiel haben wir es nun bewiesen, QuickSort hat eine CC von 8, siehe Abb 2, ist aber auch am effektivsten!

Ein weiterer **String Literal Report** (SL) zeigt alle Strings oder magic words innerhalb von Subprogrammen, die vor allem bei der Internationalisierung zum Einsatz gelangen. String-Ressourcen werden ja nicht in die Formulardatei gebunden; sie lassen sich isolieren, indem man Mithilfe des reservierten Worts `resourcestring` Konstanten deklariert. Das Auslagern als Ressourcen vereinfacht den Übersetzungsprozess und die Wartbarkeit, denn wer favorisiert schon „festverdrahteten" Code.

Nützlich kann auch der **Coupling Between Objects (CBO)** Report sein, der den Kopplungsgrad zwischen den Objekten aufzeigt und auf Wartungsprobleme hinweist, sofern die Zahl zu hoch ist (Im Beispiel mit 4 gerechnet). Mein Favorit ist natürlich das Gesetz von Demeter oder eben die Verletzung desselben (Violations of Demeters Law (VOD)), siehe Referat über Refactoring am Schluss. Sogar der altbekannte Shotgun Surgery (SS) ist mit von der Partie.

1.2 Visualisierung

Eine etwas abstraktere Sicht der Daten bietet das Kiviat-Diagramm (Abb. 3). Jede Metrik lässt sich repräsentieren durch die Speiche eines Rades, die aktuelle Bewertung eine Metrik durch einen Punkt auf dieser Speiche. In Abb. 3 hat das oben vorgestellte CBO wieder den Wert 4. Was oberhalb des roten Kreises erscheint, ist im kritischen Bereich.

Einige Tipps noch zum Management des Model View: Nachdem man für ein Projekt den Together-Support entfernt hat, ist es nicht möglich, die Diagrammdateien über die Projektverwaltung zu löschen. Wählen Sie dazu im Hauptmenü der IDE den Befehl `Projekt | Aus dem Projekt entfernen`. Daraufhin wird eine Liste der Diagrammdateien angezeigt. Sie können diese Dateien nun auswählen und löschen. Falls bei der Auswahl von Refactorings die Modellunterstützung deaktiviert ist, werden Sie in einer Meldung gefragt, ob Sie die Unterstützung für das Projekt aktivieren möchten.

Was mir noch ein Rätsel ist deutet auf den Unterschied UML 1.5 und UML 2.0 hin. Bei der standardmäßigen Unterstützung zieht Together immer die 1.5 Notation an, hier fehlt mir die Option, von Anfang an auf die 2.0 Notation zu wechseln. Gemäß Borland ist die 2.0 nur für Design-Projekte definiert.

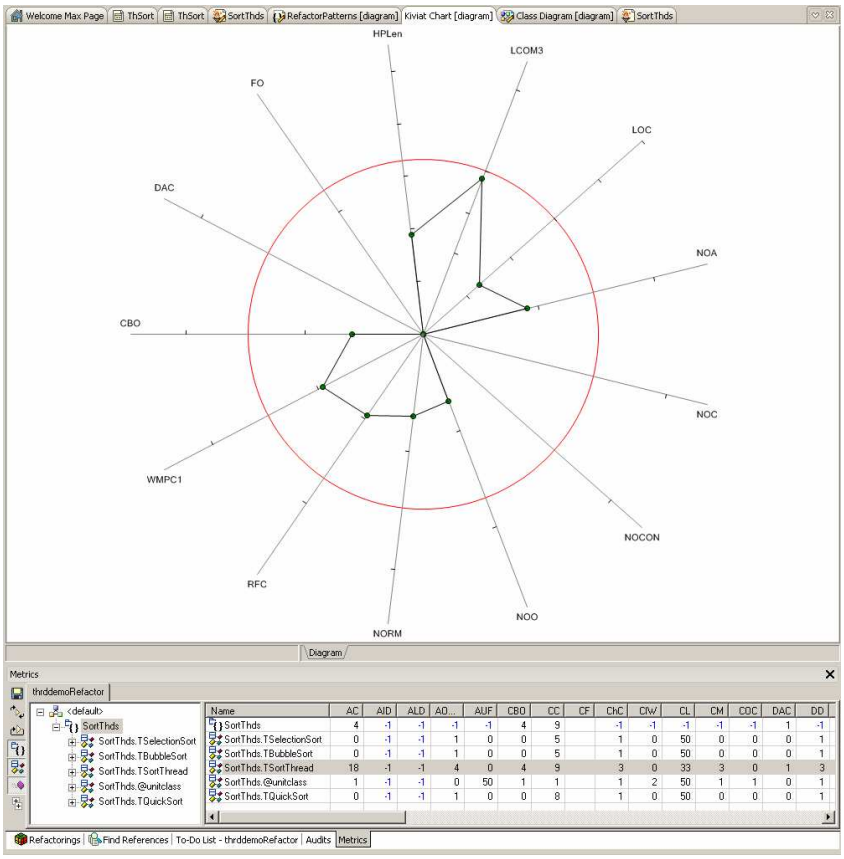


Abb. 3: Kritische Bereiche bei Kiviat

Im Weiteren lassen sich in den Modellen auch Patterns einsetzen (siehe Abb. 4). Die Änderung einer Klasse hat dann Einfluss auf die davon abhängigen Funktionen im Code, vorausgesetzt man hat ein Implementierungsprojekt. Nun was ist das wieder Neues? Bei einem Implementierungsprojekt ist nach der Definition von Together das Projekt sprachspezifisch, d.h. es beinhaltet Diagramme UND Code. Im Gegensatz zu einem reinen Design Projekt, das vor der Codegenerierung eben keine Sourcen beinhaltet. Einige Funktionen und Optionen basieren auf dieser Unterscheidung.

Sehr nützlich ist auch das Speichern von eigenen Sets, wie der Originalauszug der Hilfe bestätigt:

“Together ships with a predefined saved audit set (current.adt) and you can create your own custom sets of audits to use.” Das Gleiche gilt auch für Metriken mit der Endung .mts die in XML definiert sind und man natürlich ins Backup aufnehmen sollte.

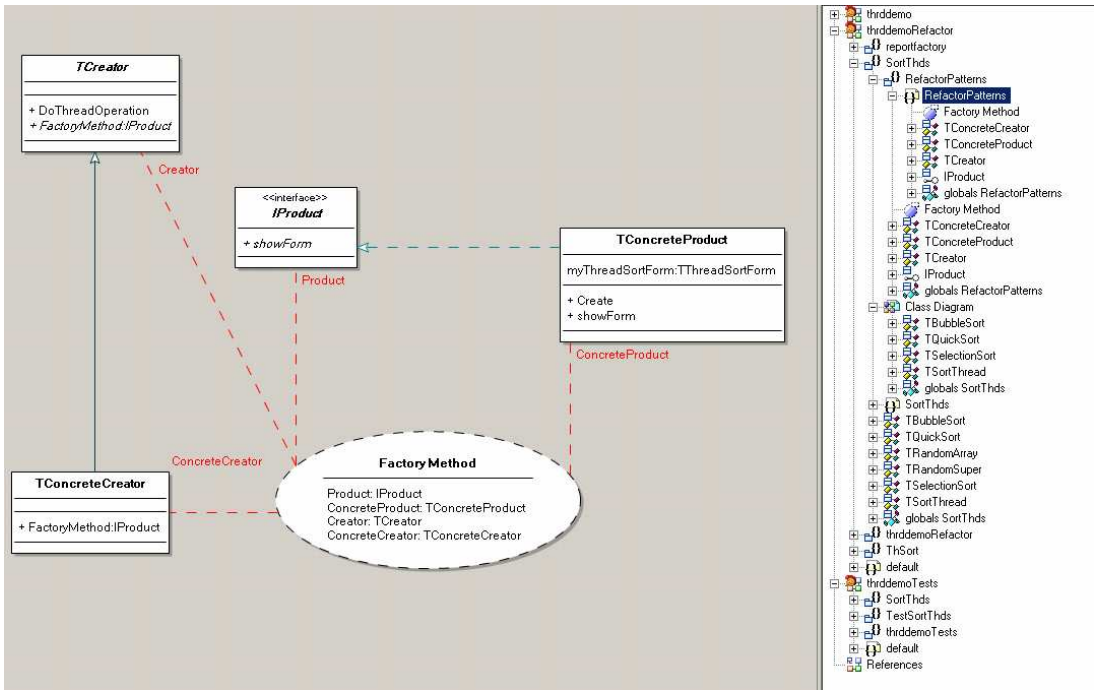


Abb. 4: Patterngenerator

Together in Delphi 2007 ist ein enormes Werkzeug. Es ist intuitiv, schnell mitinstalliert und zügig als Designtime-Package gestartet. Probleme habe ich ab und zu beim Importieren von Fremddiagrammen, z.B. von Enterprise Architect. Die Importschnittstelle bemängelt dann die Datei *EA_UML.dtd* nicht zu finden, die Lösung ist dann eine Pfadanpassung in der DOCTYPE Deklaration:

```
<!DOCTYPE XMI SYSTEM "///W:/UML_EA.dtd">
```

Hilfreich wäre endlich auch die Namenskonvention festzulegen, wobei der von Borland vorgeschlagene Kompromiss zwischen **Pascal Case** (CreateBankAccount) und Camel Case (createBankAccount) ganz gut passt. Together bedeutet im Englischen ja zusammen, will heißen, sich bei Gelegenheit von einem weiteren digitalen Kollegen über die Schultern schauen zu lassen. Damit ihr Code eleganter und leistungsfähiger wird.

Links:

Audits & Metrics
http://www.softwareschule.ch/download/refactoring_rosenheim2008_3.pdf

Namenskonvention
http://www.delphi1.com/Articles/Abbreviations.htm

Max Kleiner