



## Използване на генератори на код при създаването на готови шаблони за разработка на приложения

Даниел Дамянов

### Using Code Generators when Creating Ready-Made Templates for Application Development

Daniel Damyanov

**Abstract:** Code generation is basically about writing programs that write programs. Given today's complex code-intensive frameworks, such as Java 2 Enterprise Edition (J2EE), Microsoft's .NET and Microsoft Foundation Classes (MFC), it is becoming increasingly important that we use our skills to create programs which aid us in developing our applications. Generally speaking, the more complex the framework is, the more appealing you will find a code generation solution. Many object-oriented programming (OOP) languages lack reusability and flexibility, and require a similar code to be written repeatedly. This paper reviews the code generators that are most useful for implementation in applications for automatic code generations, their pros and cons, where they are most widely used nowadays, as well as their versatility. In Visual Studio we use the term "scaffolding" when we want to generate identities automatically. When we start a new ASP.NET project, it generates a template where we start from.

**Keywords:** generators; programming; scaffolding; active and passive code generators.

### ВЪВЕДЕНИЕ

В съвременния свят на програмиране и информационни технологии все повече програмистите се състезават с времето и спазването на поставените срокове за изпълнение на проекти. Писането на код в много случаи има повтарящи се операции, при които се губи време за описването на едни и същи функции или просто за създаването на рамката, по която ще се пишат последващите функционалности.

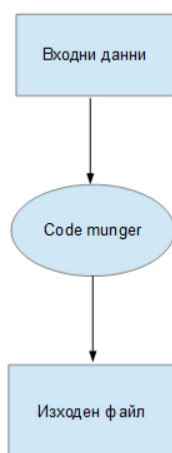
### Генератори на код

Съвременните интегрирани среди за разработка (IDE – Integrated Development Environment) съдържат генератори на код, който по зададено потребителско желание генерира шаблон, върху който програмистът да „стъпи“ и да развие проекта от някаква база. Така например в .NET и Visual Studio при избор на проект, например ASP.NET MVC, се генерира шаблон с папка с контролери, изгледи, модели и основни настройки на приложението. Отделно потребителят е улеснен при създаването на методи за основните CRUD-операции при регистрация, логин и т.н. на потребителите на самото приложение с т.нар. "Scaffolding", който генерира класове и методи, готови за употреба и интегриране в приложението. Предимствата, които такива генератори предоставят, са много, но най-големите и важни от тях са:

- качество – големите обеми код, написан ръчно, обикновено имат непостоянно качество, както и съдържат грешки в очакваните резултати. Генерирането на код от шаблони създава последователна кодова база незабавно и когато шаблоните се сменят и генераторът се стартира, корекциите на грешки или подобренията на кода се прилагат последователно в цялата кодова база;

- съвместимост – кодът, който се изгражда от генератора, е последователен в дизайна при използването на имена на променливи. Това води до предвидимост без изненади и код, който е лесен за разбиране и използване;
- повече време за проектиране – графикът за проект за генериране на код е значително по-различен от този за ръчно написан проект. В допълнение, тъй като генерирането на код компресиращ времето за определени проекти, може да се отдели повече внимание на адекватното тестване на дизайна, за да се избегне преработката надолу по веригата;
- решения за дизайн, които се открояват – бизнес правилата на високо ниво се губят в детайлите на кода за изпълнение. Генераторите на кодове използват файлове с абстрактни дефиниции, за да определят дизайна на кода, който ще се генерира. Тези файлове са много по-кратки и по-специфични от получения код. Малките изключения се открояват по-ясно във файл с дефиниция от пет реда, отколкото в получените петстотин реда код за изпълнение.

Има много начини за категоризиране на генераторите. Можем да ги класифицираме според сложността им, по начина на интегриране или по техния вход/изход. Според [4] има две основни категории за автоматично генериране на код: пасивно и активно. Генераторите на пасивни кодове изграждат кода само веднъж, тогава потребителят трябва да актуализира и поддържа кода. Най-често използваните генератори на пасивни кодове са във вид на “wizards”. Активните генератори на кодове стартират кода няколко пъти през жизнения цикъл. При тях има код, който може да се редактира от потребителите, и код, който трябва да бъде модифициран само от генератора на кодове. Активните генератори на кодове са широко споменати в литературата [6]. В [1, 3, 4, 5] изследователите дефинират шест категории активни генератори на кодове [4, р. 28]: **Code munging**: Програмата за кодове чете кода като вход и след това изгражда нов код като изход (фигура 1.). Този нов код може да бъде частичен или пълен в зависимост от дизайна на генератора. Този тип генератори е най-често срещаната форма на кодове и се използва широко. Типично тяхно приложение е автоматичното генериране на документация. Източник – източник компилатор, транскомпилатор или transpiler също могат да бъдат определени като кодове.



Фигура 1. *Code munging*

Транскомпилаторът взема код, написан на някакъв език за програмиране и го превежда в код, написан на някой друг език. Примери: C2J, JavaDoc, Jazillian, Coccinelle, Co eeScript, Dart, Haxe, TypeScript и Emscripten.

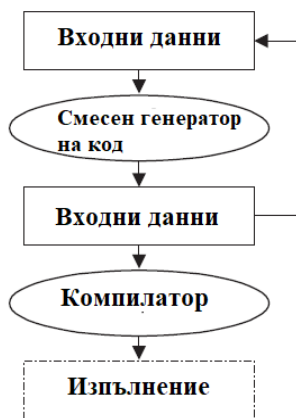
**Разширител на вграден код:** Този модел чете кода като вход и създава нов, като използва входния код за база, но с допълнителни разширения въз основа на дизайна на оригиналния.



Фигура 2. *Inline-code expander*

Започва с проектирането на нов език. Обикновено този нов език е съществуващ с някои синтактични разширения. Вграденият разширител на кода се използва, за да превърне този език в производствен код на език от високо ниво. Примери: Вградени SQL езици като SQLJ (за Java) и Pro \* C (за C). SQL може да бъде вграден в C или Java код 2 (фигура 2.).

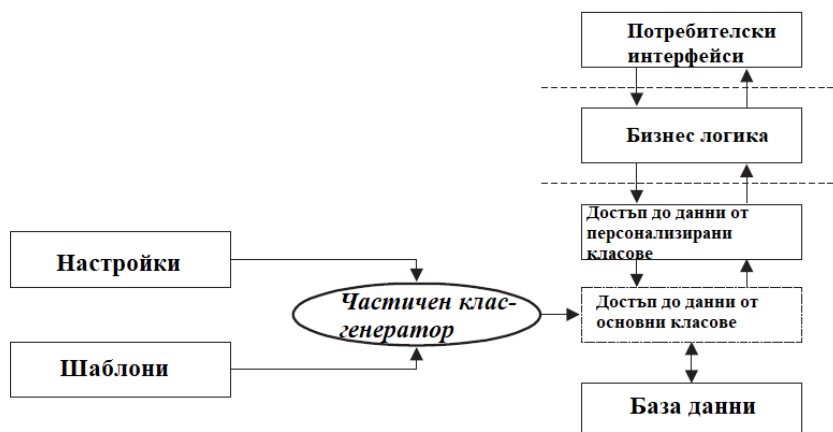
**Генератор на смесен код:** Генераторът на смесен код чете файл с изходен код и след това модифицира и заменя файла на мястото му. Това е различно от разширението с вграден код, тъй като



Фигура 3. *Смесен генератор на код (Mixed code-generator)*

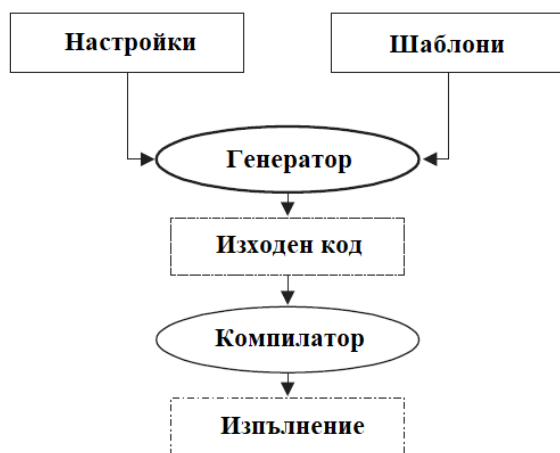
генераторът със смесен код поставя изхода на генератора обратно във входния файл. По принцип правилата за трансформация се определят с помощта на регулярни изрази (фигура 3.).

**Генератор на частични класове:** Генераторът на частичен клас приема абстрактно определение като вход вместо код (например UML диаграма на класове) и след това изгражда изходния код. След това потребителят може да го разшири чрез създаване на производни класове и добавяне на методи за завършване на дизайна. Превръщането на моделите в код се извършва чрез последователност от трансформации (фигура 4.).



Фигура 4. Генератор на частични класове

**Генератор на нива:** В този модел генераторът изгражда пълен набор от изходен код от абстрактно определение. Той има същата концепция като генератора за частичен клас. Голямата разлика между генератора на нива и частичния клас е, че в модела на подреждане генераторът изгражда целия код за подреждане. Този код е предназначен да се използва без разширения. Пример за генератор на нива, задвижван от модел, при което приложението за създаване на UML се използва съвместно с генератор и входен дефиниращ файл (Настройки, често в XML) за извеждане на едно или повече нива на система. Моделът на генератора на частични класове обаче позволява на инженера да създаде останалите производни класове, които ще завършат функционалността на нивото (фигура 5.).



Фигура 5. Генератор на нива

**Език с пълен домейн (обхват):** Тези езици са основно нови, които имат типове, синтаксис и операции, и се използват за специфичен тип проблеми. Езиците на домейни са крайни за автоматичното генериране на код, защото разработчиците трябва да напишат компилатор за всеки проблемна домейн и език.

**Метапрограма:** Метапрограмата е програма, която генерира или манипулира програмния код. Метапрограмите са описани с използване на общи конструкции от езици на високо ниво (като шаблони в C #) или друг език (например макроезик, метоезик). Метапрограмите обикновено са генерични и имат редица параметри, следователно името “generic parameter” също се използва.

Общият компонент се дефинира като софтуерен модул, позволяващ да се изберат неговите свойства до определена степен, без да се налага кодът да се пише или променя ръчно [2].

**Ниво на абстракция:** Абстракцията е основният начин за организиране на класове и групиране на подобни функционалности заедно [7]. Абстракцията скрива несъществените детайли на прилагането и набляга на информацията, която е важна за разработчика или крайния потребител в дадения контекст. Нивото на абстракция е често срещано понятие в компютърните науки.

## ЗАКЛЮЧЕНИЕ

В този доклад са разгледани основните генератори, които се използват при автоматично генериране на програмен код. Основните предимства при тяхното използване са бързото генериране на повтарящ се код и акцентирането върху други проблеми за разрешаване. Минусите, които могат да се отбележат, са в това, че програмистът ги ползва наготово и не се замисля как работят и върху тяхната логика. Така това генериране се превръща в машинално действие без анализ и вникване в кода. В днешно време при кратките срокове за изпълнение на проекти те са важни помощни средства за бързо съсредоточаване на програмиста върху съответната имплементация на програмна логика. Изследването на тези генератори има за основна цел да послужи при създаването на приложение, което при подадени входни параметри генерира краен код, лесен за интегриране в различни приложно-програмни среди – една от задачите и на моя дисертационен труд.

## ЛИТЕРАТУРА

- [1] **Bajovs, A., Nikiforova, O., Sejans, J.** 2013. Code Generation from UML Model: State of the Art and Practical Implications, *Applied Computer Systems*, 14(1), pp. 9–18. doi: <https://doi.org/10.2478/acss-2013-0002>
- [2] **Becker, M.** 2000. Generic components: a symbiosis of paradigms. In: *2nd international symposium on generative and component-based software engineering GCSE LNCS*, vol. 2177, Springer, Berlin, pp. 100–113.
- [3] **Fertalj, K., Brcic, M.** 2008. A source code generator based on UML Specification. *International journal of computers and communications*, 1(2), pp. 10–19.
- [4] **Herrington, J.** 2003. *Code Generation in Action*. Manning Publications, pp. 3.
- [5] **Hunt, A., Thomas, D.** 1999. *The Pragmatic Programmer: From Journeyman to Master*(1 ed.). USA: Addison-Wesley, pp. 27. ISBN 978-0201616224.
- [6] **Pais, R., Barros S. P., Gomes, L.** 2005. A tool for tailored code generation from Petri net models, *IEEE Conference on Emerging Technologies and Factory Automation*, Catania, pp. 857–864, doi: 10.1109/ETFA.2005.1612615.
- [7] **Stepanov, A.** 2002. Future of Abstraction. *A keynote addressed at Joint ACM Java Grande – ISCOPE 2002 Conference*, Seattle, Washington.

## ИНФОРМАЦИЯ ЗА АВТОРА

---

ас. Даниел Дамянов – докторант, специалност „Информатика“, факултет „Математика и информатика“, Великотърновски университет „Св. св. Кирил и Методий“, e-mail: [damyanovdaniel@yahoo.com](mailto:damyanovdaniel@yahoo.com)

## ABOUT THE AUTHOR

Daniel Damyanov – Lecturer, PhD Student in Informatics, Faculty of Mathematics and Informatics, St. Cyril and St. Methodius University of Veliko Tarnovo, Bulgaria, E-mail: [damyan-ovdaniel@yahoo.com](mailto:damyan-ovdaniel@yahoo.com)