 doi:10.20944/preprints202205.0398.v1

A Survey and Evaluation Model for Generative Software Development (GSD)

Mohammad Reza Besharati
Sharif University of Technology, Tehran, Iran.
besharati@ce.sharif.edu

Abstract

This survey proposed an evaluation model to analyze and examine different approaches to generativity. In addition to problem domain concepts, the following concepts were used to define this model: Complexity and complexity management, and Systematics view to achieve unified and integrated insight into disparate evaluation criteria. The research's approach to the said concepts is first introduced. Then, the evaluation model is presented.

Keywords: Generative Software Development, Code Generation, Complexty Space.

Introduction to Complexity

Definition of Complexity

Complexity is a feature of objects and systems that is normally regarded as the quality of being hard to separate, analyze, or solve (as a problem)[1]. Disparate features have been suggested for complexity in various domains, which appears that the following features are common among the majority of them[2]:

1- Size, count, and number cause complexity.

2- Relationships and the dependency of components can lead to complexity.

3- Complexity hinders perception, identification, and conclusion (Prior to complexity management)

Several other resources defined other features for complexity, which appear to be different forms of the second feature mentioned hereinabove. (Such as limitation[3]: Resource limitation, time limitation, etc., which are in fact other variants of the second feature. Lack of certainty and inability to predict the behavior of a system[4] is another alternative for counts of states of a system).

Sometimes complexity is categorized as a factor that increments the workload or controller concerns in the systems[5]. With regard to the project management domain, the project complexity is referred to as the nature,

[1] hard to separate, analyze, or solve (Bhagat, 2011)
[2] (Bhagat, 2011) (Jonathan M. Histon, 2008)
[3] (Becz, 2010)
[4] (Moses, 2004)
[5] (Jonathan M. Histon, 2008)

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

quality, and magnitude (=count) of organizational subtasks and their relationships, which are added to the organizations due to the project[1].

Some resources differentiate between these two classes of the definitions of complexity. The complexity of physical manifestation is caused by components and the relationship between them. Mental complexity is defined from the viewpoint of human users[2]. There is another definition that comprises these two ideas, which is as follows: Complexity is a feature that hinders acquiring a comprehensive and complete evaluation of a system even if we retain all information regarding the components of that system and their relationships [3]. This recent definition constituent both the mental complexity and physical manifestation of complexity.

For psychologists, complexity is a feature that prevents cognition, therefore, it provokes anxiety. In other other words, it is to understand the fact that the object or system under investigation is something other than what it was perceived or what it should be [4].

Interesting results were obtained from a study on the perception of employees of an organization from complexity. In this test, 80 individuals participated in creating an engineering product, from the senior project manager to the engineers and technical managers. They were all asked to respond to the question that "What has complicated/ complicates the module that you are creating?" Figure 1 shows the responses [5].

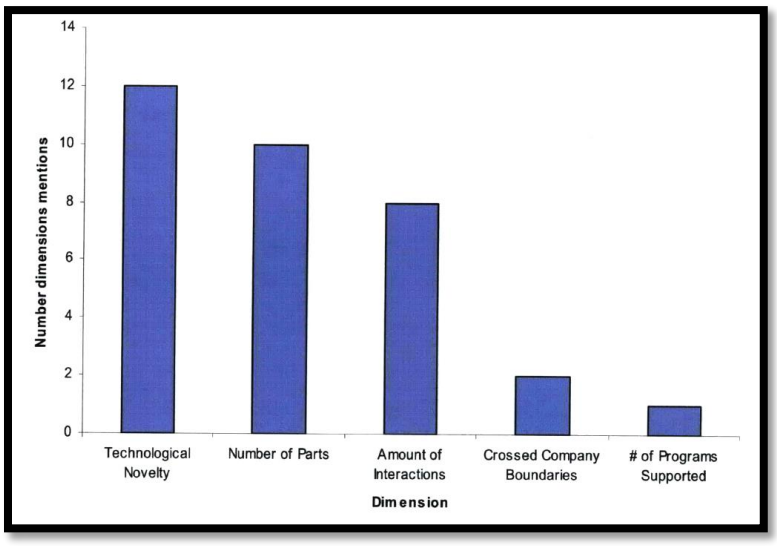

Figure 1. Responses of 80 Individuals Involved in Creating an Engineering Product in an Organization to a Question Concerning the Origin of Complexity. Reference: (Makumbe, 2008)

[1] (M.V. Tatikonda, 2000)
[2] (Bhagat, 2011)
[3] (Bhagat, 2011)
[4] (Peterson, 2002)
[5] (Makumbe, 2008)

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

In regard to the scientific scope of systems engineering, first, back in the 70s, complexity was considered to be the result of component count and the relationship between them. Recently, this field has developed a more complicated approach regarding complexity. This approach separates three separate fields of complexity i.e., structural complexity, behavioral complexity, and interface complexity [1].

The interesting point for the researcher is that the majority of software engineering texts were more concerned with structural complexity than behavioral complexity, where there was a more complicated approach toward complexity. The same applies to design patterns and software architecture, in which the main aspect is structural complexity. The question that comes into the mind is: Does the behavioral complexity is less crucial and significant than the structural complexity in the software systems, as well as software? Are were following the right paradigm? Perhaps programming and its tradition, which is based on programs with static structures, has led us to trivialize the behavioral aspect and emphasize the structural aspects of software systems. Can the relatively emerging approaches such as search-based software engineering, generative software development, and animation of formal specifications be regarded as sparks, which would one day change the dominant approach in the world of software.

Kolmogorov (1963) proposed a definition for complexity [2]. This definition is included in the wide range of structural complexity [3]. Per this definition, the complexity of a program (or algorithm) equals the length of its shortest description [4]. Thus, for the scientific use of this definition, an invariant and basic language must be proposed to describe the programs and algorithms, like a Turing machine. A crucial significance of this definition is that in the essence of its definition, it has valued the simplest and shortest description. A description that in the language of our field, will be the most abstract description [5]. Thus, a component of the system, which can be abstracted will have no impact on complexity. Meaning that to reduce complexity, we need to make it more abstract and simple. Here we will see learn that software engineers agree with Kolmogorov and abstraction is the most basic mechanism to manage complexity.

Formal Definition of Complexity and Complexity Management per the Set Theory

As already stated, there are numerous definitions of complexity. Each of which is proposed in regard to a scientific and engineering field and concerning the context of its concepts. Despite the differences, and due to the similarity of the basic concepts, a high-level and official definition of complexity is possible. Here a formal, count-based, definition of complexity will be proposed. Then, a formal definition for complexity management will be provided based on that. It should be noted that this definition is achieved through the abstraction of numerous previously mentioned definitions and it is similar to the concept of cardinality in the set theory.

[1] (Makumbe, 2008) (Moses, 2004)
[2] (Kolmogorov, 1963)
[3] (Moses, 2004)
[4] (Moses, 2004)
[5] "the Kolmogorov measure hints at a key idea, which is the use of abstractions to reduce the length of the description of the structure of a system, and thus its Kolmogorov complexity." (Moses, 2004)

 doi:10.20944/preprints202205.0398.v1

Formal Definition of Complexity

Before defining complexity, we need to define something that complexity must be calculated for. Taking into account that today the set theory is accepted as a basis for mathematics and abstraction[1], thus, the concept of set is selected as the most basic, construct, structure and concept.

A set is defined as a collection of objects. If abstraction and object are regarded as predefined and basic concepts, thus, the concept of a set can be defined based on them. On the contrary, if a set is regarded as a predefined and basic concept, thus, abstraction and object can be fined based on it. The first approach, i.e., considering abstraction and object as basis appears to be more natural to the author.

Definition 1: The complexity of a set equals the number of its members.

Definition 2: A -N set is simple if its complexity is lower than or equal to N.

Accordingly, this definition is similar to the definition of cardinality in the set theory. If the concept of abstraction is regarded as presumption and base, then, the concept of "the number of members", which was used in the above definition, will be defined at least for countable and finite sets. This concept can be developed as more compatible and well-defined for more complex structures, such as uncountable and infinite sets, which was carried out in the set theory.

Per the standpoint of this article and the approach adopted by the author concerning complexity and complexity management, we shall merely deal with the countable and finite sets. Hence, defining complexity for uncountable and infinite sets is unnecessary [2].

Relationship of Complexity, Abstraction, and Set

Therefore, complexity can be defined per set. On the contrary and following the above definition, if complexity is presented, there should be a corresponding basic set. Thus, it can be concluded that the existence of a set indicates the complexity and complexity indicates the existence of a set. Given that, the two concepts of the existence of complexity and set are equivalent, in a way, and equal from our viewpoint.

Every different abstraction of an object, a system, or an external reality leads to a different, and even

single-member, set[3]. Every set can be regarded as correspondence to an abstraction. Thus, per this study, the two concepts of abstraction and set are equivalent and equal.

[1] The fundamental objects in mathematics are sets and their constituent elements. (Hwang, 2008)
[2] However, it can be pointed out that a definition of complexity can be achieved for uncountable and infinite sets, upon applying abstraction multiple times, or abstraction of a set of abstractions. As in the set theory, cardinality calculated the uncountable and infinite sets in this way.
[3] A set is an abstraction of the informal concept of a collection of objects. (Hwang, 2008)

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

In light of that, a general yet quite crucial rule can be achieved that the concepts of the existence of complexity, abstraction, and set are all equivalent and equal concepts following the standpoint of this study.

Complexity of Calculating Complexity

It must be noted that every different abstraction of an object or system, which can be complex and contain components, will lead us to a different set. Thus, various complexities can be calculated for an object based on different abstractions. In addition, the output of abstraction can be the input of another abstraction. Each abstraction, even a single-member one, defines a set. Even though the definition proposed for complexity appears to be slightly simple, calculating complexity depends on our views regarding the system under investigation, plus it is complex and contains numerous counts with regard to the abovesaid reasons. Every different view leads to a different abstraction of the system[1], and as a result, different calculations for complexity.

The important point is that this article is not seeking to calculate complexity, and it is more concerned with detecting a complexity, rather than calculating it.

Formal Definition of Complexity Management

The definitions of complexity management were provided earlier. The concept of relationship is a basic concept in all of these definitions. In general, complexity management is relating these two complexities to each other. Following the official definition of complexity, which was mentioned earlier, complexity corresponds to the count of a set.

Consider the set A with M members. And set B with N members. To define the relationship between these two sets, first, the relationship of M members of the set A must be defined. However, for us, the relationship is an abstract and simple concept.

Thus, it appears that the most basic level of complexity management is establishing a relationship between a complex set and a simple set. Per definition 2, simplicity is a relative concept and abound must be considered for it with regard to every application and context. Thus, a broader definition is provided:

Definition 3: Establishing every relationship between M-member set A and N-member set B, is regarded as management of complexity on them.

Properties of Complexity Management

First: Complexity management is a Holonic concept. Meaning that when A and B sets are subject to complexity management, both their internal structure (i.e., members) and external structure are important (the relationship between these two sets is defined and the external structure of both sets is observed). Thus, every set in that unit is both a whole (with members) and a component (one part of the relationship[2].

[1] (Grady Booch, 2007)
[2] (Mella, 2009)

 doi:10.20944/preprints202205.0398.v1

Second: This concept can be used as a Fractal. Since a mechanism due to which two sets are related, can be applied to the internal members of every set to establish internal relationships inside that set. Given there is no presumption regarding the level of abstraction of each side of the relationship and it can be reapplied at every level[1].

Third: Every complexity management can be viewed as a constructive operator, in constructive mathematics, which takes to operands: The sides of a relationship (two sets and two complexity) are operands, and the output of this operator is a new complex space: Two previous sets together with the third set, which is the result of establishing a relationship between two previous sets. Thus, this definition of complexity management completely matches the type theory, and consequently, it is well-defined.

Fourth: Without abstraction, there will be no sets, and without sets, complexity management cannot be defined. On the other hand, the concept of relating is in fact a type of abstraction. Thus, it can be argued that the most basic pattern and mechanism of complexity management is an abstraction. Furthermore, it can be concluded that management of each complexity requires a new complexity, in form of a number of abstractions.

[1] The main characteristic of fractals is self-similarity, implying recursion, pattern-inside-of-pattern. Mandelbrot's sets display self-similarity, because they not only produce detail at finer scales but also produce details with certain constant proportions or ratios, though they are not identical. (A. Tharumarajah, 1998)

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Ivy: An assessment system to assess generative approaches

In this section, an assessment system is defined to assess generative approaches.

Theoretical Basis of Ivy

Assessment is the result of analyzing complexity. The generative approaches resulted in complexity spaces. Thus, per different analyses regarding this complexity space, there are numerous assessments concerning the generative approaches.

On the other hand, this complexity space was created by generative systems. The foundation and smallest unit of this space of complexity are systems. Thus, what we need to assess has the same quality as a system (in the general sense that includes the approaches). Thus, all types of analysis possible for a system, such as analyzing properties, efficiency, structural analysis, analyzing goals, etc., can be used to assess the complexity space of the generative approaches.

On the other hand. To avoid merely abstract analysis and to observe the problem orientation, which is the main demand of engineering and technical approaches, it is necessary to define a problem for the assessment system and construct an assessment system to help solve this problem. Meaning that the results of the assessment, should not be remained merely abstract classifications and based on the field of generation concepts. It must be deployed to other fields of engineering to find its scientific and operational value.

Thus, two main demands and two values are defined for the assessment system:

- Ordering the complexity space of generative approaches to acquire a better cognition and compare these approaches (theoretical purpose)
- Deploying the concepts of the field of generation to other software engineering fields with a problem-solving approach (practical purpose)

However, the following is the problem that the assessment system is seeking to solve: **Which generative approach is more appropriate for a project in a special situation?**

In other words, the results of the assessment must help us to deploy the space of generative approaches to the space of project situations. A common type of architecture for structuring the space of the project situations is the situational factors in the situational methodology engineering, which was proposed by Harmsen and is the basis for assessment in our system.

Thus, in the conceptual architecture of ivy system assessment (the following image), on the one hand, there is the complexity space of generative approaches and on the other hand, the complexity space of the situational methodology factors. Various types of analysis are used on systems to acquire an understanding of the generative approaches. Thus, this understanding is the basis to construct a map and transition between the complexity space of generative approaches and the situation factors' space.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

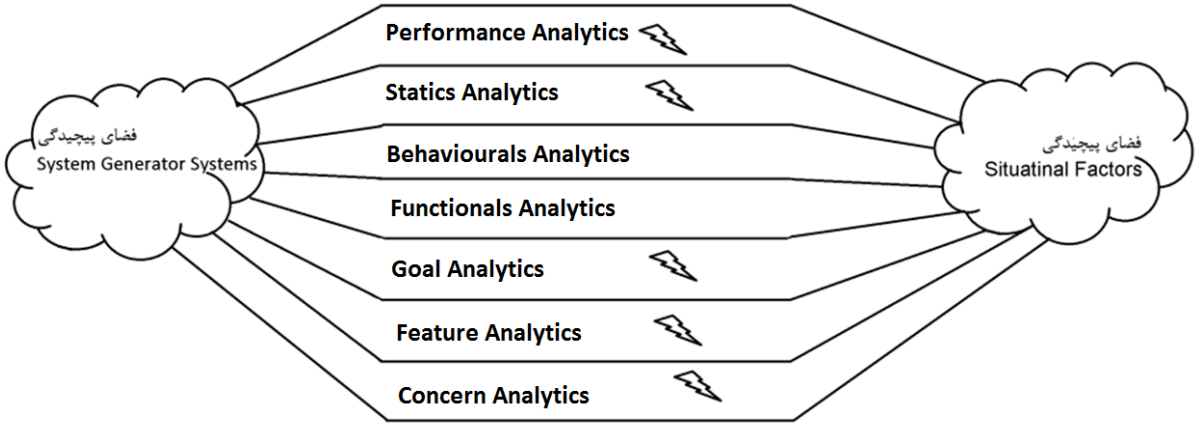

Figure 2. Conceptual Architecture of Ivy Assessment System, Marked Analyses Are Used in This Project.

Upon applying the ivy system to this complexity space (which constituents two complexity subspaces on the right and on the left), the map between these two spaces is created. Meaning that: The complexities are constantly analyzed. The result of this analysis is a semantic construction, which is constantly completed. With regard to this analysis and based on it, the map is synthesized and created. In fact, analysis is a process that transforms the complexity clouds on the right and left sides of the above figure into semantic structures (with a Systematics view). While creating these maps, these structures are developed to fill the gap between the right and left subspaces.

To solve the problems of generative mapping to project situations uses a structural approach and accomplishes its practical purpose.

To acquire its theoretical purpose (i.e., ordering the space of generative approaches), ivy uses a systematics view and approach toward the generative space. Meaning that views each of the generative approaches as a generative system and applies the analyses used for systems on them. The results of these analyses are an understanding of the system under analysis, which is created as a semantic construction. Thus, it gives a structuralist view to the analysis.

In the theoretical term, the proposition and map can be constructed so tiny that the minimum loss of meaning can occur in the course of this relationship. In fact, the analysis can be developed and detailed such that a detailed map is synthesized that can provide a plain proposition without any gap from the space of the generative concepts to the space of situation factors.

Since ivy has a constructive view of the analysis, its analyses result in a structure. The map between two left and right subspaces is a structure as well. Thus, this paradigmatic common context (view everything as a structure), enables us to reach a good gap-free level and the meanings can deploy from one subspace to another without any gaps.

However, in practice, this study provided the reader with all its knowledge in this regard. It is hoped to provide a broader and more complete description of the capacities of the ivy assessment system in future studies.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

This system is called ivy since it operates like ivy and can deploy mass phenomena (=complex) together as a construct.

The foundation of the ivy assessment system is based on the following concepts:

- Complexity, Complexity Space, and Complexity Management Approach
- Systematics approach and using various analyses on systems for complexity management
- Structuralism in analysis (=analysis, is a soft and constructive construct).
- Problem-oriented Approach
- Constructive approach for problem-solving and map creating

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Situation Factors

Numerous situation factors are mentioned for project situations in the situational method engineering literature. Following the experimental studies, the majority of these situation factors are regarded as important by project managers and included in their decisions, either, explicitly or implicitly. However, the large number of situation factors and their permutation can cause complexity in the SME process. This, effort were carried out to find the most effective factors and focus on them. Harmsen played a crucial role in forming the SME paradigm, plus, they considered the selection and classification by Hoof (below figure) to be appropriate and referred to it. He designated this limited number of factors as man situation factors.

Environment	- management commitment - importance of the information system - impact of the information system - amount of resistance and conflict - scarcity of available resources - stability and formality of the environment - knowledge and experience of the users
Project organisation	- skill of the project team - size of the project team - quality of information planning - interfaces with other projects and systems - dependency on other projects and parties
Project	- clarity and stability of project goal - quality of the specifications - size of the project - complexity of the project - novelty of the project
Constraints	- (contractual) agreement with the customer - project goal - established standards - technical restrictions

Figure 5. Main Situation Factor Determined by Hoof (ac cited by Harmsen, 1997).

It should be noted that situation factors are not merely methods to construct the complexity space of project situations, and there are other approaches (such as Aspects of Situational Methods and Characterizations Approach) as well.

On account of the limitations of the research and researcher, this study, emphasizes merely project class (5 out of 21 factors) from the abovesaid factors. Since these five factors in the project class, enjoy the highest relationship with the generative concept and automatic software programming.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Concluding Approaches to Generativity

Considering the issues stated in chapters one and two, the generative approaches in the following general classes:

1- Krzysztof Czarnecki's approach to generative
2- Model-Driven Architecture (MDA)
3- Emergence-and Evolution-based Approaches
    a. Generative Patterns per Alexander Style
    b. Human-oriented Approaches
        i. Agile Development Engine
        ii. Code Ecosystems
    c. Evolutionary and Genetic Programming Approaches
4- Formal Methods-based Approaches
    a. Constructive Approaches
    b. Animation-based Constructive
    c. Refinement-based Constructive

Definition of Generative and Criteria of Assessment

- **Complexity**

  Complexity is a set of components. The size of complexity amounts to the size of its set.

- **Generativity[1] and Coefficient of Generativity[2]**

  Generativity is a process during which a generator system automatically transforms an input complexity into an output complexity. Generativity is the deployment of these two complexity space and management complexity. the ratio of output complexity to the input complexity is the coefficient of generativity of this system.

  Generativity Coefficient = Output complexity size /input complexity size

Complexity-Based Criteria

- **Generativity complexity**

  Generativity complexity is a function comprising three components: Input complexity, generative process complexity (=dynamic complexity of the generative system), and generative system complexity (=static complexity of the generative system)

[1] Generative Software Development (GSD)
[2] Generativity

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Generativity complexity = F (input complexity, generative process complexity, generative system complexity)

- **Efficiency of Generative System**

  The efficiency of a generative system is defined per the following relation:

  The efficiency of generative system= output complexity size / generative complexity size

- **Performance of Generative System**

  Performance of generative system= size of output complexities of outputs under generative system management

- **Effectiveness**

  Effectiveness = size of output complexities of outputs under generative system management/size of complexity of total space

- **Generativity Speed**

  Generativity speed= Output complexity size /generative process complexity size

- **Reusability**
  Reusability coefficient= Reusable complexity size in output / complexity size output

- **Constructivity**

  Constructivity coefficient = 1 reusability coefficient

- **Concreteness of a Complexity**

  This supervising criterion is defined regarding a base level of abstraction. There are several base components at this base level, which are the constructive building blocks in the construction process.

  Concreteness is defined as follows:

  Concreteness= Number of base components at the structure of complexity, which are not one of its components. / Complexity Size

Criteria for Analyzing Properties

- Tool-based
- Formal method-based
- Flexibility in customization of output
- Flexibility in output softness

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

- Scalability
- Transformation type and quality
    - Refinement-based
    - Composition-based
    - Configuration-based
    - Being Hybrid
- Quality of CRUD permissions on input complexity
- Feedback-based
- Biased or unbiased: The majority of generative methods use generativity to produce a final target system in form of output. In case the essence of this goal is predetermined, it will be biased generativity. In case a specific method or direction is not specified for the process of construction and generativity in the generative system, then, it will be unbiased generativity.
- Human Interference: Some generativity systems operate automatically and independently from humans. Some other types are automatic, yet human resource is a component of the generative system.
- Level of Automaticity: Fully automated, semi-automated
- Application Domain of Generative System: Limited to a specific domain, general
- Output syntax expressivity
- Input syntax expressivity
- Construction capability (output semantic surjection)
- Input semantic surjection

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Assessment of Generative Approaches

Assessment Per Complexity-Based Criteria

The complexity-based criteria are defined to have the operationalization feature and can be calculated for concrete systems. However, some of these criteria cannot be calculated on the general system and general approaches. In such cases, a general estimation is regarded as sufficient instead of a precise calculation.

Sometimes in the assessment of an approach, several levels of values are mentioned. It means that in different samples of this approach, the level value of the criteria is different. The most common type of this value level is stated in the simplest of systems of each criterion.

Table 1. Definition of Generativity Coefficient Criterion

Criterion Name	Value Domain	Value Level
Generativity Coefficient	[0,infinity)	Smaller than 1: **Reducer** Close to 1: **Identical** Larger than 1 and smaller than 10: **Enhancer** Larger than 10: **Extremely Enhancer**

Table 2. Assessment of Generativity Coefficient Criterion

	Automatic Programming	Krzysztof Czarnecki's approach To Generativity	Approaches MDA-based	Generative Patterns Per Alexander Style	Development Engine Agile	Code Ecosystems	Evolutionary and Genetic Programming	Approaches based on Formal Methods
Status of Generativity Coefficient Criterion	Enhancer	Identical or Enhancer	Identical, Enhancer, Reducer[1]	Extremely Enhancer or Enhancer	Extremely Enhancer	Extremely Enhancer	Extremely Enhancer[2], Enhancer, Identical, Reducer[3]	Reducer[4], Identical, Enhancer

[1] In reverse transactions, this coefficient is usually reducer.
[2] For instance in content generation in the field of game development
[3] For instance, in the inductive programming approach that uses genetic programming, several samples should be given to the system as inputs for a very simple program. Thus, the level of generativity coefficient of a system becomes a reducer.
[4] Sometimes a large volume of formal descriptions is required to formally explain the essence of the problem and system. However, at the same time, the target system might be very simple,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Table 3. Definition of Generativity Coefficient Criterion

Criterion Name		Value Domain	Value Levels (Input for Calculation: Other Complexities of Environment)
Generativity Complexity	Input Complexity	[0,infinity)	Zero: **Nothing** At a lower level in regard to other complexities in the environment: **Low** At a medium level in regard to other complexities in the environment: **Medium** At a higher level in regard to other complexities in the environment: **High**
	Generativity Complexity Process	[0,infinity)	Zero: **Nothing** At a lower level in regard to other complexities in the environment: **Low** At a medium level in regard to other complexities in the environment: **Medium** At a higher level in regard to other complexities in the environment: **High**
	Complexity of Generative System	[0,infinity)	Zero: **Nothing** At a lower level in regard to other complexities in the environment: **Low** At a medium level in regard to other complexities in the environment: **Medium** At a higher level in regard to other complexities in the environment: **High**

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Table 4. Assessment of Generativity Complexity Criterion

	Automatic Programming	Krzysztof Czarnecki's approach To Generativity	Approaches MDA-based	Generative Patterns Per Alexander Style	Development Engine Agile	Code Ecosystems	Evolutionary and Genetic Programming	Approaches based on Formal Methods
Status of Input Complexity Criterion	**Medium, Low**	**Medium**[1], **High**[2], **Low**[3]	**Medium**[4] **High**	**Zero**[5], **Low**	**Low**	**Low**	Supervising several complexities of the environment: **Low**[6] Supervising others: **Medium to High**[7]	**Medium, High**

[1] At DSL-Generative approaches (=based on DSL) the status is approximately similar to automatic programming. Since the idea and even the method are similar to a higher level compiler in form of a DSL compiler.
[2] This is related to cases in which the pieces or map knowledge are received input.
[3] When there is a map based on configuration between the domain of problem and domain.
[4] In the approaches based on MDA, the models are achieved gradually and based on each other. Model complexity is based on each other's magnitude (normally three times each other as a map from one level to the other).
[5] One of the main ideas of generative pattern per Alexander's Style, is autonomy in change, development, and generativity. In this approach, systems are classified into three static, dynamic, and more dynamic. More dynamic systems enjoy dynamism without input, autonomously, and endogenous.
[6] Input complexity in regard to generativity complexity process (= number of executing construct algorithm loops in the evolutionary programs) is very very low.
[7] Input complexity in regard to output is high. (= sometimes the output of a program is simple and mass input of test samples).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Table 4. Assessment of Generativity Complexity Criterion (brought forward)

	Automatic Programming	Krzysztof Czarnecki's approach To Generativity	Approaches MDA-based	Generative Patterns Per Alexander Style	Development Engine Agile	Code Ecosystems	Evolutionary and Genetic Programming	Approaches based on Formal Methods
Status of Generativity Complexity Process Criterion	**Low, Medium**	**Low, Medium**	**Low, Medium**	**Low, Medium**[1]	**High**	**Medium**[2], **High**[3]	**High**	**High**
Criterion Status **Complexity Generative System**	**Medium, Low**	**Medium, High**[4]	**Low, Medium**	**Medium**[5]	**Low, Medium**	**High**[6]	**Low**	**Low**[7]

[1] It is interesting that in Alexander's approach, there is an extremely enhancer generativity, and at the same time, low process complexity. Since in this approach, the generativity of a phenomenon is consistent and consistent emergence-based in the course of a system's life, plus it is not a transaction with cross-sectional prosperity. In Alexander's approach to the generative patterns, there is a type of balance in all sections of a system in terms of structural, operational, etc. complexity.

[2] The structural complexities in the systems of the code ecosystem are extremely high. Since they are ecosystem and mass. Therefore, there might be an ecosystem that might have been established a few months ago (= meaning a low or medium generativity complexity process between ecosystems), however, they might be comprehensive in terms of mass development and the number of projects available in the ecosystem. (= High structural complexity)

[3] Using ecosystems for generativity requires a long-term approach to deliver results. Since the mass complexity of a system should be used to provide a mass and abundant market of products.

[4] Especially if pieces and map knowledge are provided from the problems domain to the response domain inside the system and internally, and the input is not provided. Similar to what is observed in the majority of software product lines (which use the Krzysztof Czarnecki's approach to generativity).

[5] In Alexander's approach, the system itself is responsible for generativity. Therefore it depends on itself, and with medium complexity of the environment, the complexity of the system becomes generative.

[6] High, amounting to the high complexity of an ecosystem

[7] Constructing a generative formal system is costly and requires precision, as well as trained human resources. However, the result is usually a small set of fully calculated orders, rules, and regulations. A generative formal system is normally small, yet is it fully calculated. It can be argued that the definition space of these systems is complex in a negative sense and it is simple in a positive sense (such simple that is accepted by mathematic abstraction). However, other systems usually create a small negative space, plus a large space of semantic propositions, which are neither negative nor positive (not specified). The majority of errors appear in the space of unspecified semantic propositions.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Table 5. Definition of Three Other Criteria of Complexity

Criterion Name	Value Domain	Value Level
Reusability Coefficient	[0,1]	Zero: Zero Extremely lower than 0.5: Low About 0.5: Medium Extremely higher than 0.5: High
Constructivity Coefficient (Reusability Coefficient -1)	[0,1]	Zero: Zero Extremely lower than 0.5: Low About 0.5: Medium Extremely higher than 0.5: High
Concreteness Output	[0,1]	Zero: Zero Extremely lower than 0.5: Low About 0.5: Medium Extremely higher than 0.5: High

Table 5. Assessment of Three Other Criteria of Complexity

	Automatic Programming	Krzysztof Czarnecki's approach To Generativity	Approaches MDA-based	Generative Patterns Per Alexander Style	Development Engine Agile	Code Ecosystems	Evolutionary and Genetic Programming	Approaches based on Formal Methods
Status of Reusability Coefficient Criterion	**Low**[1]	**High, Medium**[2]	**Low**[3], **Medium**	**Zero**[4], **Low**	**Low, Medium**	**Low, Medium**	**Low**	**Low**
Status of Constructivity Coefficient Criterion	**High**	**Low, Medium**	**High, Medium**	**High**	**High, Medium**	**High, Medium**	**High**	**High**
Criterion Status **Concreteness** Output	**High**	**Low**[5], **Medium**	**Medium, High**	**High**	**High**	**High**	**High**	**High**

[1] The foundation of outputs is included in the compiler system, thus, it is subject to reusability. However, the combination of these foundations is constructive. This, reusability is not zero.

[2] In case mapping is carried out from the problem domain space to solution space via transformations, then, the construction operations are possible. It should be taken into account that in Krzysztof Czarnecki's approach, mapping is carried out to determine the deployment of components and not their construct. Thus, the value of a novel construct will not exceed the value of reusability.

[3] Levels of MDA do not force reusability. In the approach based on the MDA transformation, a novel construct can be crucially important.

[4] If the use of the system's definition is not regarded as reusability.

[5] In regard to mapping with a configuration-based approach, the basic output components are coarse (= such as components, services, active libraries, etc.)

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Mapping Complexity-Based Criteria to Situation Factors

Table 6. Effect of Situation Factors on Generativity Complexity-Based Criteria

	Generativity Coefficient	Input Complexity	Complexity of Generative System	Generativity Complexity Process	Reusability Coefficient	Constructivity Coefficient	Concreteness Output
Clarity and stability of project goal	+	–[1]		+[2]	+ (With regard to Selection of Problem-solving Approach)	+ (With regard to Selection of Problem-solving Approach)	
Quality of the Specification	+	–	–	–	+ (With regard to Selection of Problem-solving Approach)	+ (With regard to Selection of Problem-solving Approach)	
Size of the project	–[3]	+	+	+	+	–	[4]–
Complexity of the project	–	+	+	+	+	–	–
Novelty of the project	–				–	+	

[1] Autonomy in a project is when the goals are clear. In most cases, it reduces the need for input.
[2] The generative system seeks to increase the complexity of the generativity process to acquire higher levels of generativity coefficient.
[3] Currently, generative approaches are not quite capable of participating in projects at the same pace of project development. From the 80s, it was believed that the fourth and fifth generations of programming languages will provide higher levels of the automation process. It was a realistic perspective, however, it has not happened yet.
[4] In such cases, modularity is used to manage the complexity of the construction process (from components, services, aspects, etc.) It usually results in a lack of output concreteness.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Assessment on the Basis of Property Analysis

Table 7. Assessment on the basis of Properties

	Automatic Programming	Krzysztof Czarnecki's approach To Generativity	Approaches MDA-based	Generative Patterns Per Alexander Style	Development Engine Agile	Code Ecosystems	Evolutionary and Genetic Programming	Approaches based on Formal Methods
Tool-based	**Yes**	**Yes**	**Not Necessarily**	**Not Necessarily**	**Not Necessarily But strongly recommended.**	**Not Necessarily**	**Yes**	**Not Necessarily**
Formal method-based	**Not Necessarily**	**No**	**Not Necessarily**	**No**[1]	**No**	**No**	**No**	**Yes**
Flexibility in customization of output	**Medium**	**High**	**Medium, High**	**Low Or Even Nothing**	**High**	**Medium**	**Low**	**Medium**
Flexibility in output softness	**Medium, High**[2]	**Medium, Low**[3]	**High, Medium**	**Low**	**High**	**High**[4]	**Medium, High**[5]	**Low or Even Nothing**[6]
Scalability	**Yes**	**Yes**	**Yes**	**No**	**No**[7]	**Yes**	**No**	**No**
Type of Transformation: Refinement-based	**No**	**Not Necessarily**	**Not Necessarily**	**No**	**No**	**No**	**Not Necessarily**	**Yes**
Type of Transformation: Composition-based	**Yes**	**Not Necessarily**	**Not Necessarily**	**Yes**	**No**	**No**	**Yes**	**Not Necessarily**
Type of Transformation: Configuration-based	**Yes**[8]	**Not Necessarily**	**Not Necessarily**	**No**	**No**	**No**	**No**	**No**
Quality of CRUD permissions on input complexity	**R**	**RUD**	**CRUD**	**CRUD**	**CRUD**	**CRUD**	**CRUD**	**R**
Feedback-based	**No**	**No**	**Not Necessarily**	**Yes**[9]	**Yes**	**Yes**	**Yes**	**No**

[1] It enjoys its special structural view.
[2] Usually the output is in form of code and can be understood by the programmer.
[3] For Krzysztof Czarnecki, a large part of the process is not supposed to be constructed as case changes.
[4] Normally, these ecosystems are formed based on free/libre and open-source software (FLOSS).
[5] Output in form of codes.
[6] Since output change without going through the process of creating generative, can cause flaws in the health and accuracy of the output code. One of the goals of formalism is to guarantee or stabilize the accuracy of output code.
[7] Scalability of Agile methods still remains controversial.
[8] For example in Template Programming
[9] Since it showed emergence-based consistent generativity.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Table 7. Assessment on the basis of Properties(brought forward)

	Automatic Programming	Krzysztof Czarnecki's approach To Generativity	Approaches MDA-based	Generative Patterns Per Alexander Style	Development Engine Agile	Code Ecosystems	Evolutionary and Genetic Programming	Approaches based on Formal Methods
Biased or unbiased	**Totally Biased**	**Totally Biased**	**Totally Biased**[1]	**Unbiased**	**Almost Biased & Almost Unbiased**	**Almost Biased & Almost Unbiased**	**Almost Biased & Almost Unbiased**	**Totally Biased**
Human Interference	**Not Acceptable.**	**Acceptable**	**Acceptable**	**Not Acceptable.** [2]	**Necessary.**	**Necessary.**	**Acceptable.**	**Necessary.**
Automaticity of Generative System	**High**	**Not Necessary (High or Low)**	**Not Necessary (High or Low)**	**High**	**High is Desirable.**[3]	**High**	**High**	**Medium**[4] **Low**
Domain of Application of Target System	**General**	**General**	**General**	**Special**	**Special**	**General**	**Special**	**General but Special in Practice**
Output syntax expressivity	**High**	**High**	**High**	**Limited**	**High**	**High**	**High but Limited in Practice**[5]	**High**
Input syntax expressivity	**Low**	**Low**	**Not Necessary (High or Low)**	**Limited**	**High**[6]	**Limited**[7]	**Limited**	**High**
Construct power (output semantic surjection)	**Limited**	**Limited**	**Not Necessary**	**Limited**	**Medium**	**High**	**Extremely Limited**	**High**
Input semantic surjection	**Limited**	**High**[8]	**Not Necessary**	**Limited**	**Limited**	**Limited**	**Limited**	**High**[9]

[1] It is normally the case and there are exceptions as well. For instance, in *MD software regarding Self-Adaptability
[2] Unless humans are regarded as part of the generative system.
[3] By automaticity, it means that the generative system is automated. Agile approaches emphasize the self-organization of teams and linearity of the complexity of the human component of the creation process in form of agile teams. Thus, the most crucial goal regarding the formation of Agile teams is their independent operation without imposing a management load.
[4] In the majority of official approaches, several tools are provided concerning automatization. However, full automatization is not achieved.
[5] To express special levels of outputs, there might be the need for extremely complex processing, which is practically impossible,
[6] Usually includes natural language and documents of natural language.
[7] Input complexity to a code ecosystem comprises the signals of human and commercial relationships, which are sent to that ecosystem. In this case, the input complexity of the structure is limited.
[8] Products, concepts, and models of the problem domain must be placed inside the system or in their input. Therefore, the input of the generative system per Krzysztof Czarnecki style is a big input. This input might be provided in form of a Repository inside a generative system. However, regardless of where these inputs are placed, they are necessary for Krzysztof Czarnecki's approach.
[9] Parts of the semantics of the target system are transferred to the formal approaches in their input, which supervises the essence of target system instead of its quality.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Mapping Property Criteria to Situation Factors

Table 8. Suitability of Property to Situation

	Clarity and stability of project goal	Quality of the Specification	Size of the project	Complexity of the project	Novelty of the project
Tool-based	+	+	+	+	+
Formal method-based	+	+	-	-	-
Flexibility in customization of output			+	+	+
Flexibility in output softness			+	+	+
Scalability			+	+	
Type of Transformation: Refinement-based	+	+	-	-	+[1]
Type of Transformation: Composition-based			+	+	
Type of Transformation: Configuration-based	+	+	+	+	-
Quality of CRUD permissions on input complexity		**Permission to change might lead to danger**			+
Feedback-based			+	+	+

[1] Since refinement helps clear and step-by-step analysis, as well as clear creativity in the creation process.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

	Clarity and stability of project goal	Quality of the Specification	Size of the project	Complexity of the project	Novelty of the project
Biased	+	+			-
Unbiased	-	-	+ (Emergence-based Complexity Management)	+ (Emergence-based Complexity Management)	+
Human Interference					+
High Level of Automaticity of Generative System	+	+	+	+	-
General Domain of Target System Application					+[1]
Special Domain of Target System Application		+	+[2]	+[3]	+[4]
Output syntax expressivity			+	+	+
Input syntax expressivity	+	+			
Construct power (output semantic surjection)			+	+	+
Input semantic surjection	+	+			

[1] The majority of innovative systems interface with the fields of application. For these target systems, general generative tools are more appropriate.
[2] Specialization needs complexity management.
[3] Specialization needs complexity management.
[4] It might reduce the time-to-market for innovative systems.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 May 2022 doi:10.20944/preprints202205.0398.v1

Conclusion

Having investigated the generative approach space, the following overall classification was proposed for these approaches.

1- Automatic Programming
2- Krzysztof Czarnecki's approach to generative[1]
3- MDA-based approaches
4- Emergence-and Evolution-based Approaches
    a. Generative Patterns per Alexander Style
    b. Human-oriented Approaches
        i. Agile Development Engine
        ii. Code Ecosystems
    c. Evolutionary and Genetic Programming Approaches
5- Formal Methods-based Approaches
    a. Constructive Approaches
    b. Animation-based Approaches
    c. Refinement-based Approaches

Following the analysis, it was pointed out that generative approaches are quite different both in terms of the criteria related to complexity and in terms of property analysis. This difference can be employed to select an appropriate generative approach for the situation of a project. Accordingly, it was considered the practical goal in the ivy assessment system to become pragmatic. Meaning that it creates a map between the results from analyzing the generative approaches and the situation factors in methodology engineering.

For the author, the most important difference between various generative approaches is the difference in the generativity problem-solving method. Several methods seek to acquire generativity via the reusability of the previously constructed products. Therefore, constructing products is not a part of the problem for them. The following approaches are included in this class:

- Such as the majority of Automatic Programming approaches
- Krzysztof Czarnecki's approach to generativity
- Some MDA-based approaches

Some other methods try to solve the generativity problem by constructing somehow generative and autonomous products. Meaning that they try to construct an order instead of reusing an order. For instance, via artificial intelligence, logical inference, or upon relying on emergence phenomenon, complexity engineering, or evolutionary and genetic programming. The following approaches are included in this class:

- A few Automatic Programming approaches
- Some MDA-based approaches
- Emergence-and Evolution-based Approaches
    - Generative Patterns per Alexander Style

[1] Generative Software Development

    - Human-oriented Approaches
        - Agile Development Engine
        - Code Ecosystems
    - Evolutionary and Genetic Programming Approaches
- Formal Methods-based Approaches
    - Constructive Approaches
    - Animation-based Approaches
    - Refinement-based Approaches

Meaning that the main choice and problem, which should be carried out by a generative approach, is to select a generativity method. One of these methods:

- A method for constructing an order
- A method for reusability of an order
- And a hybrid solution

When a hybrid solution is selected, the main problem is the deployment problem: How to deploy the reused orders to new orders. It requires an analysis operation (= determining or extracting a structure for a complexity) on the reused orders or a consistence supply of levels of structuralism on reusable orders.

References

Andrea Arcuri, X. Y. (2010). Co-evolutionary automatic programming for software development. *Information Sciences*.

Ang Chen, D. B. (2006). Generative Business Process Prototyping Framework. *Seventeenth IEEE International Workshop on Rapid System Prototyping*, (pp. 140-168).

Bergadano, F. (1996). *Inductive Logic Programming: from machine learning to software engineering.* The MIT Press.

Czarnecki, K. (2004). Overview of Generative Software Development. *Unconventional Programming Paradigms (UPP).* Paris, France.

Damien Cassou, E. B. (2011). Leveraging Software Architectures to Guide and Verify the Development of Sense/Compute/Control Applications. *the 33rd International Conference on Software Engineering (ICSE'11).*

Francisco Montero Simarro, J. V. (2010). Generative Pattern-Based Design of User Interfaces. *1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems .* Berlin, Germany: ACM.

Goerigk, W. (2011). Model Driven Development of Distributed Business Applications. *17th GI/ITG Conference on Communication in Distributed Systems (KiVS 2011. 17*, pp. 211-213. Schloss Dagstuhl--Leibniz-Zentrum fuer Informatik.

Goldstein, J. (1999). Emergence as a Construct: History and Issues. *Emergence, A Journal of Complexity Issues in Organizations and Management, 1*(1).

Harmsen, A. F. (1997). *Situational method engineering.*

Hoverd, T. (2008). *Generative Patterns of Softwar: Qualifying Dissertation.* University of York.

Kitzelmann, E. (2010). Inductive programming: A survey of program synthesis techniques. *Approaches and Applications of Inductive Programming*, 50-73.

Makumbe, P. O. (2008). *Globally distributed product development : role of complexity in the what, where and how.* Massachusetts Institute of Technology.

Markus Völter, J. B. (2004). Patterns for Model-Driven Software-Development. *EuroPLoP.*

Michael. (2003). *Grammatical evolution: evolutionary automatic programming in an arbitrary language.* Springer.

Michael Zapf, T. W. (2007). Offline emergence engineering for agent societies. *the Fifth European Workshop on Multi-Agent Systems EUMAS.*

Michalewicz, Z. (1996). Evolutionary Programming and Genetic Programming. In *Genetic Algorithms+ Data Structures= Evolution Programs* (pp. 283-287). Springer.

Mihaela Ulieru, R. D. (2011). Emergent engineering: a radical paradigm shift. *International Journal of Autonomous and Adaptive Communications Systems*, 39-60.

Regina Frei, G. D. (2011). Advances in complexity engineering. *International Journal of Bio-Inspired Computation*, 199-212.

S. MacDonald, D. S. (2002). Generative Design Patterns. *17th IEEE International Conference on Automated Software Engineering (ASE 2002).*

S. Meliá, J. G. (2010). Architectural and Technological Variability in Rich Internet Applications. *IEEE Internet Computing*, 24-32.

Susan Stepney, F. A. (2006). Engineering emergence. *11th IEEE International Conference on Engineering of Complex Computer Systems(ICECCS06).*

Wąsowski, A. (2004). Automatic Generation of Program Families by Model Restrictions. *Software Product Lines: Third International Conference, SPLC04.*

Xu Ke, K. S. (2007). COMDES-II: A Component-Based Framework for Generative Development of Distributed Real-Time Control Systems. *the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications.* IEEE Computer Society.

 doi:10.20944/preprints202205.0398.v1

Other Bibliography

Speakman, John. "Evolving source code: Object oriented genetic programming in. net core." (2019): 16-19.

Bordis, Tabea, Tobias Runge, David Schultz, and Ina Schaefer. "Family-based and product-based development of correct-by-construction software product lines." *Journal of Computer Languages* (2022): 101119.

Karmakar, Rahul. "Formal Verification Techniques: A Comparative Analysis for Critical System Design." In *International Conference on Intelligent Systems Design and Applications*, pp. 93-102. Springer, Cham, 2022.

Merouani, Hemza, Fateh Boutekkouk, and Imad Merouani. "UML/event-B-based modelling and verification of the car cruise control system." *International Journal of Computer Aided Engineering and Technology* 16, no. 1 (2022): 14-39.

Zhong, Maosheng, Gen Liu, Hongwei Li, Jiangling Kuang, Jinshan Zeng, and Mingwen Wang. "CodeGen-Test: An Automatic Code Generation Model Integrating Program Test Information." *arXiv preprint arXiv:2202.07612* (2022).