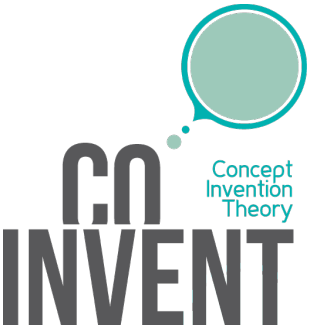


D5.2
Evaluation criteria for automated serendipity

Authors	Joseph Corneli, Alison Pease, Simon Colton
Reviewers	Tarek Besold

Grant agreement no.	611553
Project acronym	COINVENT - Concept Invention Theory
Date	September 1, 2015
Distribution	PU/RE/CO

Disclaimer

The project COINVENT acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open Grant number 611553.

Abstract

We adapt the Standardised Procedure for Evaluating Creative Systems (SPECS), a high-level, customisable evaluation strategy that was devised to judge the creativity of computational systems to turn our definition into a set of evaluation guidelines. We pilot our framework by examining the degree of serendipity of existing and hypothetical computational systems from three domains: evolutionary computing, recommender systems, and automated programming. In each of the examples, the evaluation framework shows ways in which the system's potential for serendipity could be enhanced. For future work, we propose an approach to evaluation that would build on the ideas of this report but more fully embed evaluation within the system itself.

Keyword list: **serendipity**, **evaluation**, **computational creativity**, **autonomous systems**

Changes

Version	Date	Author	Changes
0.1	01.08.15	Joseph Corneli	Creation
0.2	01.09.15	Joseph Corneli	Initial version for review
0.3	28.09.15	Joseph Corneli	Response to reviewer comments

Executive Summary

This document expands and revises the preliminary of evaluation criteria presented in D5.1. In order to be self-contained, a succinct summary our model of serendipity is presented in an appendix. The model shows the inputs, outputs, and processing steps we are prepared to call "serendipity", at a suitably abstract level for broad application.

The present document focuses on three case studies that show how the model can be used to evaluate the potential for serendipity in computational systems. A definition of serendipitous processing is introduced. A central feature of our definition is that the relevant processing steps are not guaranteed in advance.

We adapt the Standardised Procedure for Evaluating Creative Systems (SPECS), a high-level, customisable evaluation strategy that was devised to judge the creativity of computational systems to turn our definition into a set of evaluation guidelines. We pilot our framework by examining the degree of serendipity of existing and hypothetical computational systems from three domains: evolutionary computing, recommender systems, and automated programming. In each of the examples, the evaluation framework shows ways in which the system's potential for serendipity could be enhanced.

This appears to be the first proposed method for evaluating computational serendipity. For future work, we propose an approach to evaluation that would build on the ideas of this report but more fully embed evaluation within the system itself.

Contents

1 Introduction

This document expands and revises the preliminary of evaluation criteria presented in Section 5.2 ("Using SPECS to evaluate computational serendipity") of D5.1 ("Computational account of serendipity"). It follows the conceptual framework for serendipity introduced in that deliverable. In order to be self-contained, a succinct summary our model of serendipity is presented in Appendix A of the current document. The model shows the inputs, outputs, and processing steps we are prepared to call "serendipity", at a suitably abstract level.

The present document focuses on three case studies that show how the model can be used to evaluate the potential for serendipity in computational systems. Further technical details on the third case study appear in D8.4. Full details are presented in [5]. The key elements of our conceptual framework are encapsulated in the following:

Definition of serendipity. (1) Within a system with a prepared mind, a previously uninteresting trigger arises due to circumstances that the system does not control and cannot predict, and is classified as interesting by the system; and, (2) The system uses the trigger and prior preparation, together with relevant computational processing, networking, and experimental techniques, to obtain a novel result that is evaluated favourably by the system or by external sources.

The several components of this definition – the **focus shift** that facilitates the move from (1) to (2), the **trigger** itself, the **prepared mind**, the **bridge**, the **result** – are left generic, so that researchers can apply them to a range of systems. For instance, serendipity may also arise in composite socio-technical systems. A standard spell-checking program might suggest a substitution that the user deems especially fortuitous; and we might agree that serendipity has taken place, although we would not attribute serendipity to the spell-checker. In this case, it is the user's prepared mind that forms the bridge from trigger to result. Note, however, that it should not be assumed that any system that can accommodate user interaction will directly lead to serendipity; take for example the use of a calculator, where the potential for serendipity through user interaction is minimal.

A central feature of our definition is that the relevant processing steps are not guaranteed in advance. We will refer to the **chance** that the trigger is observed by the system; **curiosity** as a measure of the likelihood that the trigger will be identified as interesting; **sagacity** as a measure of the likelihood that a bridge to the result will be found. We also need to estimate the **value** of the result. Note that "embedded evaluation" is a requisite part of our definition of serendipity, and that this is different from the kind of overall evaluation we describe here.

The factors that influence serendipitous outcomes are encouraged by a number of qualitative factors, including a **dynamic world**, exposure to **multiple contexts**, interaction with **multiple tasks**, and a system that is affected by **multiple influences**. These contributing factors are familiar from historical instances of serendipitous discovery and invention Programming computers that can make use of these features is likely to pose challenges. In the case studies, we use these factors to look at ways in which the system's potential for serendipity could be improved.

In [5], we argue that serendipity is particularly relevant for *autonomous systems*, and that the potential for serendipity goes along with the ability to *learn*, the capacity to interact in a recognisably *social* manner, and that building greater sophistication at *embedded evaluation* is a good starting point for developers to work from.

2 Using SPECS to evaluate computational serendipity

We adapt the *Standardised Procedure for Evaluating Creative Systems* (SPECS), a high-level, customisable evaluation strategy that was devised to judge the creativity of computational systems [10]. In the three step SPECS process, the evaluator defines the concepts and behaviours that signal creativity, converts this definition into clear standards, and then applies them to evaluate the target systems. We follow a slightly modified version of Jordanous's earlier evaluation guidelines, in that rather than attempt a definition and evaluation of *creativity*, we follow the three steps for *serendipity*. With similar adaptations, SPECS could be used to evaluate other system features (like "intelligence"). Note that on the understanding developed here, serendipity is only meaningfully attributed to a particular kind of process. It is not a property of a generated artefact (like novelty or usefulness), nor is it a system trait (like skill or autonomy).

Step 1: Identify a definition of serendipity that your system should satisfy to be considered serendipitous.

Definition of serendipity. (1) Within a system with a prepared mind, a previously uninteresting serendipity trigger arises due to circumstances that the system does not control, and is classified as interesting by the system; and, (2) The system uses the trigger and prior preparation, together with relevant computational processing, networking, and experimental techniques, to obtain a novel result that is evaluated favourably by the system or by external sources.

Step 2: Using Step 1, clearly state what standards you use to evaluate the serendipity of your system.

With our definition and other features of the model in mind, we propose the following standards for evaluating serendipity in computational systems. These criteria allow the evaluator to assess the degree of seredipity that is present in a given system's operation.

(A - Definitional characteristics) The system can be said to have a **prepared mind**, consisting of previous experiences, background knowledge, a store of unsolved problems, skills, expectations, readiness to learn, and (optionally) a current focus or goal. It then processes a **trigger** that is at least partially the result of factors outside of its control, including randomness or unexpected events. It classifies this trigger as interesting, constituting a **focus shift**. The system then uses reasoning techniques and/or social or otherwise externally enacted alternatives to create a **bridge** from the trigger to a result. The **result** is evaluated as useful, by the system and/or by an external source.

(B - Dimensions) Serendipity, and its various dimensions, can be present to a greater or lesser degree. If the criteria above have been met, we consider the system (and optionally, generate ratings as estimated probabilities) along several dimensions: (**a**: **chance**) how likely was this trigger to appear to the system? (**b**: **curiosity**) On a population basis, comparing similar circumstances, how likely was the trigger to be identified as interesting? (**c**: **sagacity**) On a population basis, comparing similar circumstances, how likely was it that the trigger would be turned into a result? Finally, we ask, again, comparing similar results where possible: (**d**:

value) How valuable is the result that is ultimately produced? Low likelihood $\mathbf{a} \times \mathbf{b} \times \mathbf{c}$ and high value **d** are the criteria we use to say that the event was "highly serendipitous."

(C - Factors) Finally, if the criteria from Part A are met, and if the event is deemed sufficiently serendipitous to warrant further investigation according to the criteria in Part B, then in order to deepen our qualitative understanding of the serendipitous behaviour, we ask: To what extent does the system exist in a **dynamic world**, spanning **multiple contexts**, featuring **multiple tasks**, and incorporating **multiple influences**?

Step 3: Test your serendipitous system against the standards stated in Step 2 and report the results.

We will pilot our framework by examining the degree of serendipity of existing and hypothetical computational systems in the following case studies.

Pease et al. [15] used an earlier variant the SPECS criteria to analyse three examples of potentially serendipitous behaviour: dynamic investigation problems, model generation, and poetry flowcharts. Using our updated criteria, we discuss two new examples below, and revisit poetry flowcharts in our third example, reporting on recent work. As Campbell [3] writes, "serendipity presupposes a smart mind," and these examples suggest potential directions for further work in computational intelligence.

3 Case Study: Evolutionary music improvisation

[9] reported a computational jazz improvisation system (later given the name GAmprovising [10]) that uses genetic algorithms. Reevaluating GAmprovising can shed light on the degree to which evolutionary computing can encourage computational serendipity.

GAmprovising uses genetic algorithms to evolve a population of *Improvisors*. Each Improvisor is able to randomly generate music based on various parameters such as the range of notes to be used, preferred notes, rhythmic implications around note lengths and other musical parameters, see [9]. These parameters are what define the Improvisor at any point in the system's evolution. After a cycle of evolution, each Improvisor is evaluated using a fitness function based on Ritchie's [16] formal criteria for creativity. This model relies on user-supplied ratings of the novelty and appropriateness of the music produced by the Improvisor to calculate 18 metrics that collectively indicate how creative the system is. The fittest Improvisors are used to seed a new generation of Improvisors, through crossover and mutation operations.

The GAmprovising system can be said to have a **prepared mind** through its background knowledge of what musical concepts to embed in the Improvisors and the evolutionary abilities to evolve Improvisors. A potential **serendipity trigger** comes from the combination of previous mutation and crossover operations with current user input. To be clear, in the current version of the system a human evaluator is largely responsible for the system's **focus shift**, since the user tells the system which improvisations are most valuable. [9] notes that this "introduces a fitness bottleneck." In future versions of the system, autonomous evaluation could potentially take over for the human evaluator. Once the interesting samples have been collected (from whatever source), a **bridge** is then built to new results through the creation of new Improvisors. The **results** are

the various musical improvisations produced by the fittest Improvisors (as well as, perhaps, the parameters that have been considered fittest).

The probability of encountering any particular pair of Improvisor and user evaluation is vanishingly low, given the massive dimensions of this search space. However, there will always be a highest-scoring Improviser, whose parameters will be used to seed the next round. Do we estimate the **chance** of the trigger appearing according to its uniqueness, or according to the system's attentive observation of all triggers that cross its path? Consider de Mestral's encounter with burrs: the chance of encountering burrs while out walking was high, and the details of the particular burrs that were encountered effectively irrelevant. The situation here is similar: despite their uniqueness, the trigger appearing is "high." The evolution of Improvisors captures a sense of **curiosity** about how to satisfy the musical tastes of a particular human user who identifies certain Improvisors as interesting. The **sagacity** of the system corresponds to its methods for enhancing the likelihood that the user will appreciate a given Improvisor's music (or similar music) over time. With little basis for comparison, we can only say that these two dimensions are "typical." The aim of the system is to maximise the **value** of the generated results by employing a fitness function. Indeed, the system

> "[W]*as able to produce jazz improvisations which slowly evolved from what was essentially random noise, to become more pleasing and sound more like jazz to the human evaluator's ears*" [9].

However, the very reliability of the system ultimately bears against its overall potential for serendipity. Following Step 2, Part B of the SPECS procedure, we find a likelihood measure of *high* × *moderate* × *moderate*, with outcomes of moderate value, so that the system as a whole is "not very serendipitous." Evaluating individual threads (as members of a larger population) would yield varied results, which emphasises the importance of system scoping, mentioned above. However, it would be inaccurate to simply say that successful threads are serendipitous and unsuccessful threads are unserendipitous, since that ignores components other than value. At the moment, individual threads are effectively equivalent regarding chance, curiosity, and sagacity; a thread-by-thread analysis should be deferred until there would be more to say.

This is related to other changes that would improve the global serendipity score, as the following qualitative factor analysis indicates. The GAmprovising system does operate in **dynamic world**, assuming that the user's tastes may change. A more elaborate version of the system that could cater to multiple users is not yet implemented, but would be occupied with a considerably more complex problem, spanning and integrating **multiple contexts**. The system clearly engages with **multiple tasks**, but these are largely separate, for instance, one global fitness function is used, rather than evolving a local fitness function for each user along with their ratings. **Multiple influences** are present but currently only at compile time, in the design of the fitness function, and the selection of musical parameters that can later be set. Greater dynamism in future versions of the system would be likely to increase its potential for serendipity.

4 Case Study: Next-generation recommender systems

Recommender systems are one of the primary contexts in computing where serendipity is currently discussed. Serendipity, for current recommender systems, means suggesting items to a user

that will be likely to introduce new ideas that are unexpected, but close to what the user is already interested in. As we noted, these systems mostly focus on supporting *discovery* for the user – but some architectures also seem to take account of *invention* of new methods for making recommendations, e.g. using Bayesian methods, as surveyed in [7]. In light of our working definition of serendipity, we need to distinguish serendipity on the user side from serendipity in the system itself.

Current recommendation techniques associate less popular items with high unexpectedness [8, 12], and use clustering to discover latent structures in the search space, e.g., partitioning users into clusters of common interests, or clustering users and domain objects [11, 14, 17]. But even in the Bayesian case, the system has limited autonomy. A case for giving more autonomy to recommender systems can be made, especially in complex and rapidly evolving domains where hand-tuning is cost-intensive or infeasible.

With this challenge in mind, we investigate how serendipity could be achieved on the system side. In terms of our model, current systems have at least the makings of a **prepared mind**, comprising both a user- and a domain model, both of which can be updated dynamically. User behaviour (e.g. following certain recommendations) or changes to the domain (e.g. adding a new product) may serve as a potential **trigger** that could ultimately cause the system to discover a new way to make recommendations in the future. Note, however, that it is unexpected behaviour in aggregate, rather than a one-off event, that is likely to provide grounds for a **focus shift**. A **bridge** to a new kind of recommendation could be created by looking at exceptional patterns as they appear over time. For instance, new elements may have been introduced into the domain that do not cluster well, and clusters may appear in the user model that do not have obvious connections between them. A new recommendation strategy serves the organisational mission would be a serendipitous **result** for the system.

The system has only imperfect knowledge of user preferences and interests. At least relative to current recommender systems, the **chance** of noticing some particular pattern in user behaviour seems quite low. The urge to make recommendations specifically for the purposes of finding out more about users could be described as **curiosity**. Such recommendations may work to the detriment of other metrics over the short term. In principle, the system's curiosity could be set as a parameter, depending on how much coherence is permitted to suffer for the sake of gaining new knowledge. Measures of **sagacity** would relate to the system's ability to develop useful experiments and draw sensible inferences from user behaviour. For example, the system would have to select the best time to initiate an A/B test. A significant amount of programming would have to be invested in order to make this sort of judgement call autonomously, so such systems are understandably rare. The **value** of recommendation strategies can be measured in terms of traditional business metrics or other organisational objectives. In this case, we compute a likelihood measure of $low \times variable \times low$, with outcomes of potentially high value, so that such a system is "potentially highly serendipitous."

Recommender systems have to cope with a **dynamic world** of changing user preferences and a changing collection of items to recommend. A dynamic environment which exhibits some degree of regularity represents a precondition for useful A/B testing. The system's **multiple contexts** include the user model, the domain model, as well as an evolving model of its own organisation. A system matching the description here would have **multiple tasks**: making useful recommendations, generating new experiments to learn about users, and improving its models. In order to

make effective decisions, a system would have to avail itself of **multiple influences** related to experimental design, psychology, and domain understanding.

5 Case Study: Automated flowchart assembly

Here we consider the design of a contemporary experiment with the FloWr flowcharting framework [4]. FloWr is a user interface for creating and runnable flowcharts, built of small modules called ProcessNodes. In day-to-day use, FloWr can be viewed as a visual programming environment. However, it can also be invoked programmatically, on the Java Virtual Machine, or with any language using a new web API. The goals of FloWr are both to be a user friendly tool for co-creativity, and to be an autonomous *Flowchart Writer*. Our experiment targets the latter scenario, assembling available ProcessNodes into flowcharts automatically. This can be viewed as a simple example of automated programming.

In the backend, FloWr's flowcharts are stored as scripts. These detail the names of the involved nodes together with their (input) parameters and (output) variable settings. Connections between nodes are established when one node's input parameter references the output variable of another node. Inputs and outputs have constraints. For instance, the `WordSenseCategoriser` node has a `stringsToCategorise` parameter, which is seeded with an ArrayList of strings. The node produces useful output only when these strings can be parsed as as a space-separated list of words. The node's `requiredSense` parameter needs to be seeded with a string that represents exactly one of the 57 British National Corpus Part of Speech tags. Given constraints of this nature, the first challenge in automated flowchart assembly is to match inputs to outputs correctly, and to make sure that all required inputs are satisfied.

In our current experiment, the system's potential **triggers** result from trial and error with flowchart assembly. Some valid combinations of nodes will produce results, and some will not. Due to the dynamically changing environment (e.g., updates to data sources like Twitter) some flowcharts that did not produce results earlier may unexpectedly begin to produce results. The system's **prepared mind** lies in a distributed knowledge base provided by ProcessNodes, showing the constraints on their inputs and outputs, and in the global history of successful and unsuccessful combinations. The system will not try combinations that it knows cannot produce results, but it will try novel combinations and may retry earlier flowchart specimens that have the chance to become viable. Turning a collection of nodes for which no known working combination existed into a working flowchart is an occasion for a **focus shift**: what made this particular combination work? Is there a pattern that could be exploited in the future? However, it may be that no broader pattern can be found, other than the fact that the combination works. Successful combinations and any further inferences are stored, and referred to in future runs. The **bridge** to the next set of results is accordingly found by informed trial and error. In these early experiments, the basic **result** the system is aiming to achieve is simply to generate a new combination of nodes that can fit together and that generate non-empty output. Subsequent versions of the system may have more detailed evaluation functions, setting a higher bar. For example, a future version of the system could be tuned to search for flowcharts that generate poetry, as we discuss in [6].

The **chance** of finding a novel successful combination of nodes is fairly low, as this depends on both the output from certain nodes, and in terms the combinatorial search strategy itself. Compared to humans users of FloWr, the search process is exceptionally **curious**, since it tries many

combinations programmatically. Remembering viable combinations and avoiding combinations that are known not to work presents only a modest degree of **sagacity**. At the moment, the system's criterion for attributing **value** is simply that the combination of nodes generates non-empty output; however an external evaluator is not likely to judge these combinations as useful. The associated likelihood score is $low \times low \times high$, which is favourable, however, until there is a more discriminating way to judge value, the attribution of serendipity to any particular run may be premature. One fairly obvious route would be to attribute value to explanatory heuristics, rather than generated texts; this would require increased sagacity on the part of the system as well.

The **dynamic world** the system operates in is dynamic in two ways: first, in the straightforward sense that some of the input sources, like Twitter, are changing; and also in the sense that the system's knowledge of successful and unsuccessful node combinations changes over time. The current version of the system does not seem to deal with **multiple contexts**; even though we have broken the experiment into separate sub-populations to constrain the search, these do not interact. However, in a future version of the system, interaction between different heuristically-driven search processes would be possible, and could produce more unexpected results. Along these lines, as more goals are added, the system could more readily be seen to have **multiple tasks**. For instance, one search process could look for narrative outlines to structure a poem with, and another process could look for lines or stanzas to fill out that outline. As for **multiple influences**, the population of ProcessNodes will constrain (and, as more nodes are added, extend) the possible strategies for assembling flowcharts.

6 Summary and Conclusions

Table 1 summarises how the condition, components, dimensions and factors in our model of serendipity appear in an evolutionary music system, in hypothetical "next-generation" recommender systems, and in our current work on a flowchart-assembly system. Each of the case studies shows clear potential for serendipity. There are also clear ways in which the measure of serendipity could be enhanced.

1. A future version of the evolutionary music system would be more convincingly sagacious if it could evaluate works without user intervention. It might also be able to tailor its fitness function to the individual user. More broadly, interaction between the system's tasks and more dynamism in its influences would help differentiate individual threads or system runs, and some elements of this population might be more serendipitous than others.

2. The next-generation recommender systems we've envisioned need to be able to make inferences from aggregate user behaviour. This points to long-term considerations that go beyond the unique serendipitous event. How "curious" should these systems be? One obvious criterion is that short-term value should be allowed to suffer as long as expected value is still higher.

3. The flowchart assembly process would need more stringent, and more meaningful, criteria for value before third-party observers would be likely to attribute serendipity to the system. In addition to raising challenges for autonomous evaluation (as in the evolutionary music

Evolutionary music	Next-gen. recommenders	Flowchart assembly
Condition		
Focus shift		
Driven by (currently, human) evaluation of samples	Unexpected behaviour in the aggregate	Find a pattern to explain a successful combination of nodes
Components		
Trigger		
Previous evolutionary steps, in combination with user input	Input from user behaviour	Trial and error in combinatorial search
Prepared mind		
Musical knowledge, evolution mechanisms	Through user/domain model	Constraints on node inputs and outputs; history of successes and failures
Bridge		
Newly-evolved Improvisors	Elements identified outside clusters	Try novel combinations
Result		
Music generated by the fittest Improvisors	Dependent on organisation goals	Non-empty or more highly qualified output
Dimensions		
Chance		
Looking for rare gems in a huge search space	Imperfect knowledge of user preferences and behaviour	Changing state of the outside world; random selection of nodes to try
Curiosity		
Aiming to have a particular user take note of an Improvisor	Making unusual recommendations	Search for novel combinations
Sagacity		
Enhance user appreciation of Improvisor over time, using a fitness function	Update recommendation model after user behaviour	Don't try things known not to work; consider variations on successful patterns
Value		
Via fitness function (as a proxy measure of creativity)	Per business metrics/objectives	Currently "non-empty results"; more interesting evaluation functions possible
Factors		
Dynamic world		
Changes in the user tastes	As precondition for testing system's influences on user behaviour	Changing data sources and growing domain knowledge
Multiple contexts		
Multiple users' opinions would change what the system is curious about and require greater sagacity	User model, domain model, model of its own behaviour	Interaction between different heuristic search processes would increase unexpectedness
Multiple tasks		
Evolve Improvisors, generate music, collect user input, carry out fitness calculations	Make recommendations, learn from users, update models	Generate new heuristics and new domain artefacts
Multiple influences		
Through programming of fitness function and musical parameter combinations	Experimental design, psychology, domain understanding	Learning to combine new kinds of ProcessNodes

Table 1: Summary: applying our computational serendipity model to three case studies

system case), this requirement would impose more sophisticated constaints on processing in earlier steps, which would require the system to be more sagacious.

As an overall comment on the evalation method, this appears to be the first proposed method for evaluating computational serendipity, so comparison to other methods is not currently possible. Nevertheless, some comments can be advanced. Firstly, the proposal is more of a meta-method than a specific testing procedure: as we saw in the case studies, there was some work to do in each case to sufficiently define the components of serendipity required by the model. It may be useful in certain contexts to fix more concrete circumscribed definitions and thresholds. For instance, in addition to the question, "how curious should these systems be?", a related question is how fallible should they be. Pease et al. [15] reported "willingly mak[ing] the system less effective to encourage incidents which onto which we might project the word serendipity," which is not the direction we ultimately want to go, though it serves to underscore the importance of fallibility.

In [5], we propose to create a *pattern language*, after [1], describing various plans that can be put in place to encourage serendipitous discovery and invention. In this setting, each pattern would be fallible, but the ability to learn new patterns would create a sort of guarantee against absolute failure. For example, in an apparent run of bad luck, interdepartmental meetings would not produce any new ideas, but noticing the run of bad luck, and changing behaviour accordingly would help to ensure something was learned after all. The key thing about this approach is that it more fully embeds evaluation within the system.

It is worth noting that this approach has direct bearing on conceptual blending. in [2], we noted that that the concept blending generally makes some assumptions about the form of the blend. An experimental approach in which the system asks and answers questions, and evaluates results would have to transform raw data into "strategic data", after [13, p. 506]. The guidelines in this report give an idea about how to bring about the focus shifts that can help make that happen.

References

[1] ALEXANDER, C., ISHIKAWA, S., AND SILVERSTEIN, M. *A Pattern Language: Towns, Buildings, Construction*. Center for Environmental Structure Series. Oxford University Press, Oxford, 1977.

[2] BOU, F., CORNELI, J., GÓMEZ-RAMÍREZ, D., MACLEAN, E., SMAILL, A., AND PEASE, A. The role of blending in mathematical invention. In *Proceedings of the Sixth International Conference on Computational Creativity, ICCC 2015*, S. Colton, H. Toivonen, M. Cook, and D. Ventura, Eds. 2015.

[3] CAMPBELL, W. C. Serendipity in research involving laboratory animals. *ILAR journal 46*, 4 (2005), 329–331.

[4] COLTON, S., AND CHARNLEY, J. Towards a Flowcharting System for Automated Process Invention. In *Proceedings of the Fifth International Conference on Computational Creativity* (2014), D. Ventura, S. Colton, N. Lavrac, and M. Cook, Eds.

[5] CORNELI, J., JORDANOUS, A., GUCKELSBERGER, C., PEASE, A., AND COLTON, S. Modelling serendipity in a computational context. Submitted to Minds and Machines (Oct 2015).

[6] CORNELI, J., JORDANOUS, A., SHEPPERD, R., LLANO, M. T., MISZTAL, J., COLTON, S., AND GUCKELSBERGER, C. Computational poetry workshop: Making sense of work in progress. In *Proceedings of the Sixth International Conference on Computational Creativity, ICCC 2015*, S. Colton, H. Toivonen, M. Cook, and D. Ventura, Eds. Association for Computational Creativity, 2015.

[7] GUO, S. *Bayesian Recommender Systems: Models and Algorithms*. PhD thesis, The Australian National University, 2011.

[8] HERLOCKER, J. L., KONSTAN, J. A., TERVEEN, L. G., AND RIEDL, J. T. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems 22*, 1 (Jan. 2004), 5–53.

[9] JORDANOUS, A. A fitness function for creativity in jazz improvisation and beyond. In *Proceedings of the International Conference on Computational Creativity* (Lisbon, Portugal, 2010), pp. 223–227.

[10] JORDANOUS, A. A Standardised Procedure for Evaluating Creative Systems: Computational Creativity Evaluation Based on What it is to be Creative. *Cognitive Computation 4*, 3 (2012), 246–279.

[11] KAMAHARA, J., AND ASAKAWA, T. A community-based recommendation system to reveal unexpected interests. In *Proceedings of the 11th Internationall Multimedia Modelling Conference* (2005), pp. 433–438.

[12] LU, Q., CHEN, T., ZHANG, W., YANG, D., AND YU, Y. Serendipitous Personalized Ranking for Top-N Recommendation. *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology* (Dec. 2012), 258–265.

[13] MERTON, R. K. The Bearing of Empirical Research upon the Development of Social Theory. *American Sociological Review* (1948), 505–515.

[14] ONUMA, K., TONG, H., AND FALOUTSOS, C. TANGENT: A Novel 'Surprise-me' Recommendation Algorithm. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (2009).

[15] PEASE, A., COLTON, S., RAMEZANI, R., CHARNLEY, J., AND REED, K. A Discussion on Serendipity in Creative Systems. In *Proceedings of the Fourth International Conference on Computational Creativity* (2013).

[16] RITCHIE, G. D. Some empirical criteria for attributing creativity to a computer program. *Minds and Machines 17* (2007), 67–99.

[17] ZHANG, Y. C., SÉAGHDHA, D. O., QUERCIA, D., AND JAMBOR, T. Auralist: Introducing Serendipity into Music Recommendation. In *Proceedings of the fifth ACM international conference on Web search and data mining* (2011), pp. 13–22.

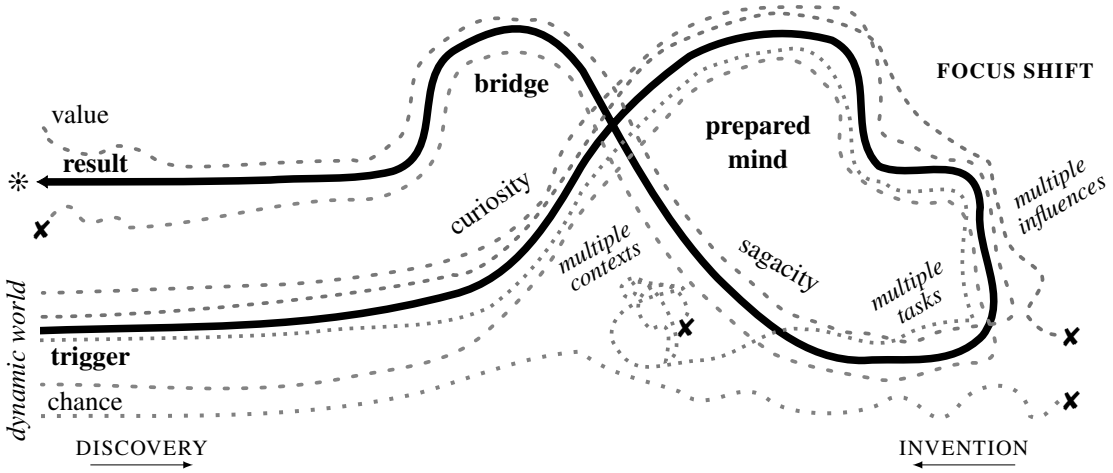
A A computational model of serendipity

Figure 1a is a heuristic map of the features of serendipity. Dashed paths ending in '✘' show some of the things that can go wrong. A serendipity trigger might not arise, or might not attract interest. If interest is aroused, a path to a useful result may not be sought, or may not be found. If a result is developed, it may turn out to be of little value. Prior experience with a related problem could be informative, but could also hamper innovation. Similarly, multiple tasks, influences, and contexts can help to foster an inventive frame of mind, but they may also be distractions.

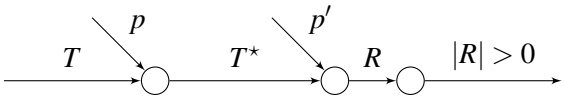
Figure 1b removes these unserendipitous paths to focus on the key features of "successful" serendipity. The **serendipity trigger** is denoted here by T. The **prepared mind** corresponds to those preparations, labeled p and p', that are relevant to the discovery and invention phases, respectively. These preparations may include training, current attitude, access to relevant knowledge sources, and so on. A **focus shift** takes place when the trigger is observed to be interesting. The now-interesting trigger is denoted $T^\star$, and is common to both the discovery and the invention phases. The **bridge** is comprised of the actions based on p' that are taken on $T^\star$ leading to the **result** R, which is ultimately given a positive evaluation.

Figure 1c expands this schematic into a sketch of the components of one possible idealised implementation of a serendipitous system. An existing *generative process* is assumed. This may be based on observations of the outside world, or it may be a purely computational process. In any case, its products are passed on to the next stage. After running this data through a feedback loop, certain aspects of the data are singled out, and marked up as "interesting." Note that this designation need not arise all at once: rather, it the outcome of a *reflective process*. In the implementation envisioned here, this process makes use of two primary functions: p_1, which notices particular aspects of the data, and p_2, which offers reflections about those aspects. Together, these functions build up a "feedback object," $T^\star$, which consists of the original data and further metadata. This is passed on to an *experimental process*, which has the task of verifying that the data is indeed interesting, and determining what it may be useful for. This is again an iterative process, relying on functions p_1' and p_2', which build a contextual understanding of the trigger by devising experiments and assessing their results. Once implications or applications have been found, a result is generated, which is passed to a final *evaluation process*, and, from there, to applications.

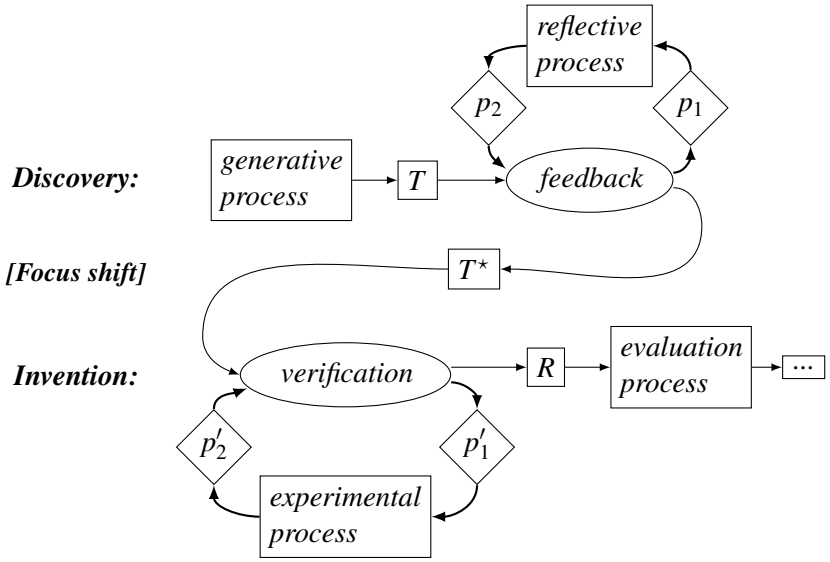
The ellipses at the end of the workflow in Figure 1c are intended to suggest that applications are open-ended; however, an important class of applications will result in changes to one or more of the system's modules, for example by expanding the knowledge base that they have available. Note that earlier components of the workflow cannot, in general, anticipate what the subsequent phases will produce or achieve. If the system's next steps could be anticipated, we would not say that the behaviour was serendipitous. In other words, serendipity does not adhere to one specific part of the system, but to its operations as a whole. Although Figures 1b and 1c treat the case of successful serendipity, as indicated in Figure 1a, each step is fallible, as is the system as a whole. Thus, for example, a trigger that has been initially tagged as interesting may prove to be fruitless. Similarly a system that implements all of the steps in Figure 1c, but that never achieves results of value does not have potential for serendipity. However, a system only produces results of high value would also be suspect, since it would indicate a tight coupling between trigger and outcome. Fallibility is a "meta-criterion" that transcends the other criteria.



(a) A heuristic map, showing serendipitous and unserendipitous outcomes



(b) A simplified process schematic, showing the key features of the model



(c) A boxes-and-arrows diagram, showing one possible implementation architecture

Figure 1: Three representations of the elements of serendipity