DOI: https://doi.org/10.15276/ict.02.2025.45
UDC 004.8:004.415.2:519.17

Multi-modal graph representations for robust anti-pattern detection in evolving codebases

Dmytro D. Kurinko[1)]
Postgraduate Student of the Department of Artificial Intelligence and Data Analysis
ORCID: https://orcid.org/0000-0001-8304-3257; dmitrykurinko@gmail.com
Viktoriia I. Kryvda[1)]
PhD, Associate Professor of the Department of Electricity and Energy Management
ORCID: https://orcid.org/0000-0002-0930-1163; kryvda@op.edu.ua
[1)] Odesa Polytechnic National University, 1, Shevchenko Ave. Odesa, 65044, Ukraine

ABSTRACT

This study examines whether multi-modal and multi-level representations enhance the reliability of code smell and anti-pattern detection in evolving polyglot software systems. A hybrid model is introduced that integrates four evidence channels – structural, semantic, metric, and evolutionary – within a unified Code Property Graph (CPG) combining AST, CFG, and PDG relations. Semantic information is obtained from pretrained code language models, while classical quality indicators (e.g., CK, McCabe/Halstead) are attached as node and edge attributes; version-control signals (e.g., churn, co-change, recency) are aggregated with time decay to emphasize recent activity. Learning proceeds hierarchically: a local encoder summarizes token-level idioms and induced graph slices; a component-level, relation-aware GNN captures cohesion/coupling and data/control-flow structure; and a project-level encoder propagates context on a component-interaction graph. Instance-wise channel gating is employed to weight modalities, thereby emphasizing source-specific and smell-specific evidence.

To support deployment under open-world conditions, selective prediction is adopted using complementary uncertainty criteria (logit energy, predictive entropy, stochastic variance), with temperature calibration to improve probability reliability and enable abstention on unfamiliar or low-confidence cases. The empirical evaluation spans Java, Kotlin, and Scala repositories under cross-project and time-aware splits; open-set tests are formed by withholding one smell class during training. Relative to rule/metric baselines, AST-GNN, text-only, and AST+Text systems, the hybrid model yields consistent improvements without increasing FPR@95TPR. Averaged over repositories, Macro-AUPRC improves by approximately 6–7 percentage points and Macro-F1 by 3–4 points over the strongest single-view baseline, with the largest gains observed for God Class and Shotgun-Surgery–like categories. Incremental CPG updates and bounded project-level propagation maintain CI/CD-compatible latency, while hierarchical attention and channel-importance scores provide reviewer-aligned explanations. The findings indicate that smells are inherently multi-signal and context-dependent, and that hierarchical, calibrated, open-set detection offers a favorable balance between accuracy and operational safety.

Keywords: Machine learning; software engineering; program analysis; graph models; static analysis; anti-pattern detection; software quality; open-set recognition

The study aims to enable early, reliable detection of code smells and design anti-patterns in rapidly evolving, polyglot software systems. Modern codebases change under continuous integration and dependency churn, where degradation emerges gradually across local idioms, structural dependencies, quantitative quality signals, and version-control history. However, most existing detectors are *single-channel* (rules/metrics, AST-only, or text-only) and *single-level* (method/file scope). As a result, they miss context-dependent cases, suffer precision–recall trade-offs, and transfer poorly across repositories and languages [1]. Project-specific thresholds drift, temporal evolution is often ignored and closed-set classifiers confidently mislabel previously unseen patterns – producing noisy CI pipelines, reviewer fatigue, and inconsistent triage. To address these shortcomings, the study aims to (i) fuse structure, semantics, metrics, and evolution into a unified representation, (ii) reason hierarchically from local code to component and project context, (iii) support open-set abstention with calibrated probabilities to avoid overconfident errors, and (iv) operate incrementally so that inference remains compatible with CI/CD latency and resource budgets [2].

The study aims to design a hybrid, multi-level model for detecting code smells and anti-patterns that improve accuracy and reliability in polyglot, evolving codebases without compromising CI/CD viability. The objectives are: (1) to integrate structural (AST+CFG+PDG), semantic (code embeddings), metric (CK, McCabe/Halstead), and evolutionary features into aified representation; (2) to enable hierarchical reasoning (local → component → project); (3) to un

develop open-set selective prediction with calibrated probabilities; (4) to support incremental processing of changes; and (5) to empirically evaluate effectiveness, transferability, calibration, and efficiency [3].

Proposed model. We cast smell detection as multi-modal, multi-level learning over a hybrid code graph that fuses structure, semantics, metrics, and evolution. Source is compiled into a Code Property Graph (CPG) unifying syntax, control, and data flow:

$$G = \left(V, E_{ast} \cup E_{cfg} \cup E_{pdg}\right), \tag{1}$$

where G is the Code Property Graph (CPG), a unified graph representation of code; V is set of graph nodes (e.g., tokens, statements, methods, classes); E_{ast} is edges from the Abstract Syntax Tree (syntactic parent/child, sibling); E_{cfg} is edges from the Control Flow Graph (execution ordering/branching); E_{pdg} is edges from the Control Flow Graph (execution ordering/branching).

On this graph, each method/class node carries four channels: (i) structural features from AST/CFG/PDG slices; (ii) semantic vectors from a pretrained code LM (e.g., CodeBERT/CodeT5) to capture naming and idioms; (iii) metric features (CK, McCabe/Halstead, size) normalized per project to reduce scale effects; and (iv) evolutionary signals mined from version control (churn, co-change, author dispersion, recency), aggregated with time decay to emphasize recent activity [4-6].

Learning proceeds hierarchically. A local encoder summarizes method-level idioms from token sequences and their induced subgraphs. These local summaries are fed to a component encoder – a relation-aware GNN over class-level CPG slices – to capture cohesion/coupling and data-/control-flow regularities. A lightweight project encoder then propagates context on a component-interaction graph (calls, imports, co-change), which is crucial for smells whose evidence is dispersed (e.g., *God Class*, *Shotgun-Surgery–like*).

Across levels, a channel-gated fusion learns instance-specific weights, so the model emphasizes the most informative evidence for each decision:

$$h = ||_{k\in\{str,sem,met,evo\}} \alpha_k W_k x_k, \qquad \alpha_k = \frac{e^{u_k^T x_k}}{\sum_j e^{u_j^T x_j}}, \tag{2}$$

where h is fused representation fed to the higher-level encoders/classifier; $k \in \{str, sem, met, evo\}$ – channel index – structure (CPG features); semantics (code embeddings); metrics (CK, McCabe/Halstead, etc.); evolution (VCS-derived features); x_k is input feature vector from channel k for the current unit (method/class/component); W_k is learnable projection matrix mapping x_k into the shared space; α_k is instance-wise gate (soft weight) for channel k; $\sum_k \alpha_k = 1$, u_k is learnable gating vector for channel k (computes the logit for α_k); j is index over all channels in the denominator (softmax normalization).

To make deployment safe in evolving repositories, the classifier is selective under open-set conditions. We combine three complementary uncertainty signals – logit energy (confidence margin), predictive entropy (class ambiguity), and stochastic variance (e.g., MC-dropout) – to decide when to abstain and route a case to human review:

$$\mathbb{I}_{abstain} = [Energy > \tau_E] \vee [H > \tau_H] \vee [V > \tau_V], \tag{3}$$

where $\mathbb{I}_{abstain} \in \{0,1\}$ is indicator that the model abstains (1) or proceeds with a label (0); H is predictive entropy of the class distribution p; V is stochastic variance of predictions across Monte-Carlo dropout samples; τ_E, τ_H, τ_V are tunable thresholds for energy, entropy, and variance, set on validation to balance coverage versus risk.

Probabilities are calibrated (temperature scaling) so CI/CD thresholds behave predictably; in practice, high scores correspond to high empirical precision [7]. The pipeline is engineered for incremental inference: only touched files are re-indexed into CPG slices, local summaries recomputed for changed methods, component states update within a bounded neighborhood, and

project propagation depth is capped (e.g., one or two hops). This preserves latency budgets while retaining the benefits of multi-signal fusion and hierarchical context. Explanations are exposed through attention maps and channel-importance scores, aligning flagged evidence with reviewer intuition and supporting triage and learning-from-feedback in real projects [8].

Data and experimental design. Evaluation was conducted on polyglot JVM corpora (Java, Kotlin, Scala) collected from public repositories, comprising tens of thousands of classes and hundreds of thousands of methods (see Table 1). Smell/anti-pattern labels were derived from a consensus of established detectors (rules/metrics) with targeted manual audits on sampled cases. Targets are multi-label: *Long Method (LM)*, *God Class (GC)*, *Feature Envy (FE)*, *Data Class (DC)*, *Shotgun-Surgery–like (SS)*, and *No-smell*.

Table 1. **Datasets summary (per language)**

Language	Repositories	Classes	Methods	Avg. LOC / method	Time span (first→last commit)
Java	45	38120	214350	13.7	2014–2024
Kotlin	18	9480	55260	14.9	2017–2024
Scala	12	6210	36840	15.1	2015–2024
Total	**75**	**53810**	**306450**	–	–

To reduce leakage, we used cross-project and time-aware splits: training on earlier commits/projects and validation/test on later, distinct projects. Open-set conditions were simulated by withholding one or more smell classes during training and revealing them only at test time; cross-language transfer was assessed by training on Java and testing on Kotlin/Scala.

Baselines included rule/metric detectors, AST-GNN, text-only, and AST+Text hybrids. Ablations removed evolution features, restricted reasoning to the local level only, or disabled selective prediction. Primary metrics were AUPRC and F1; safety/robustness used FPR@95TPR, AUROC-OSR, TNR@TPR (open-set), and ECE (calibration). Hyperparameters were tuned on validation with class-balanced batches and loss weighting. Operational viability was measured via inference latency in CI-like scenarios (small/medium/large pull requests) using incremental CPG updates and capped project-level propagation depth.

Results and discussion. At the language level, the hybrid model consistently leads across Java, Kotlin, and Scala (see Fig. 1 and Fig. 2).

Table 2. **Average Macro-AUPRC (mean over repositories)**

Language	Rules/Metrics	AST-GNN	Text-only	AST+Text	Hybrid (Ours)
Java	0.490	0.588	0.570	0.610	0.676
Kotlin	0.507	0.592	0.581	0.625	0.698
Scala	0.508	0.599	0.595	0.629	0.696

Averaged over repositories, Macro-AUPRC gains of the Hybrid system over the strongest single-view baseline (AST+Text) range from ~5-7 percentage points on Java and Scala to ~6-7 points on Kotlin, with corresponding Macro-F1 improvements of ~3-4 points (see Table 2 and Table 3 for exact values).

Table 3. **Average Macro-F1 (mean over repositories)**

Language	Rules/Metrics	AST-GNN	Text-only	AST+Text	Hybrid (Ours)
Java	0.500	0.568	0.550	0.591	0.656
Kotlin	0.500	0.571	0.559	0.609	0.682
Scala	0.503	0.579	0.577	0.609	0.678

These margins are stable despite different language idioms and corpus sizes, indicating that hierarchical aggregation and channel-gated fusion generalize beyond per-project specifics.

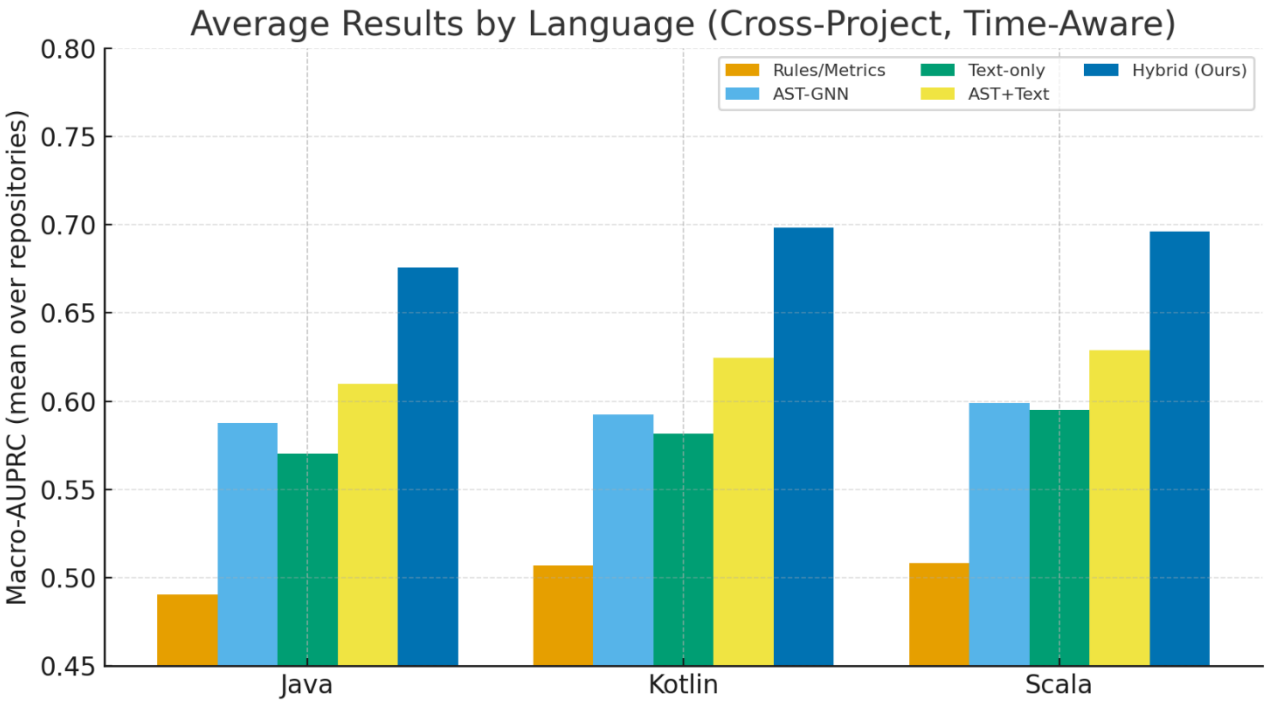

Fig. 1. **Average Macro-AURPC by Language (Cross-Project, Time-Aware)**

Two effects explain the pattern. First, languages whose projects exhibit richer inter-component interaction (notably several Java and Scala repositories) benefit most from the project-level encoder: evidence for God Class and Shotgun-Surgery–like smells is dispersed, so propagating context across the interaction graph reduces false positives at high recall.

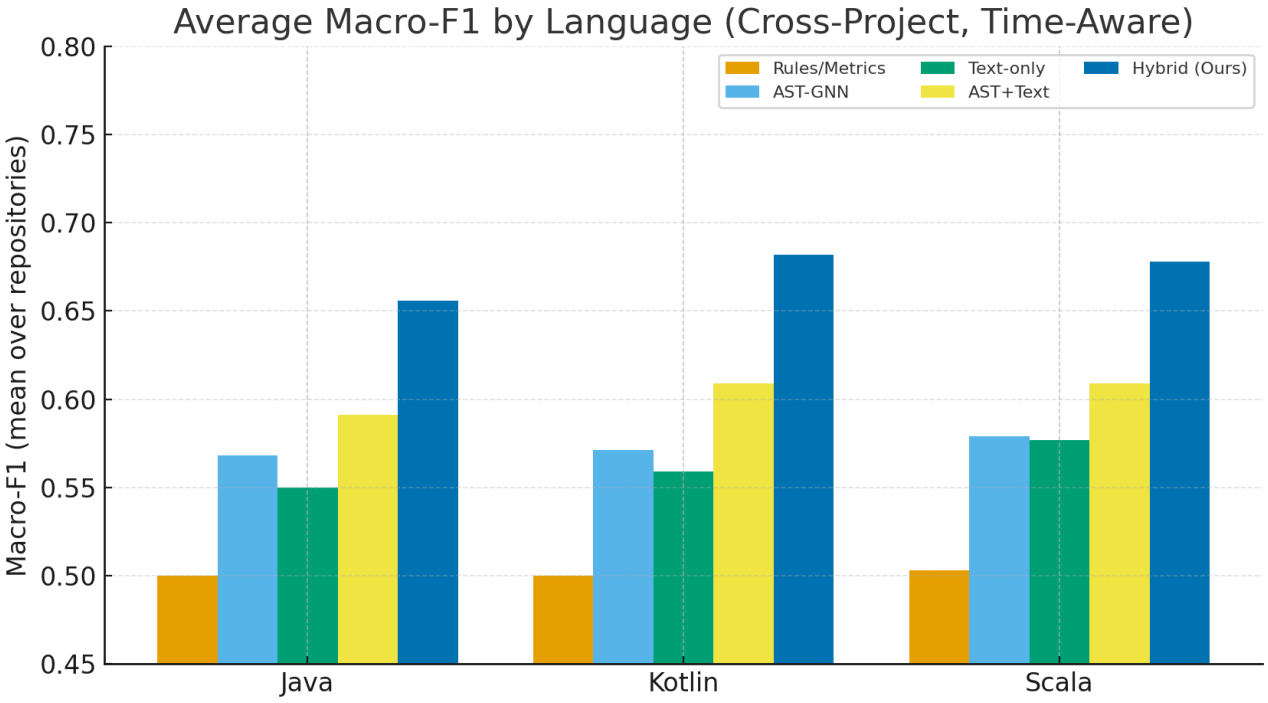

Fig. 2. **Average Macro-F1 by Language (Cross-Project, Time-Aware)**

Second, repositories with higher churn and stronger co-change signals improve under the evolution channel, which adds discriminative temporal features missing from structural or semantic views alone. Kotlin shows slightly smaller absolute gains, consistent with more uniform idioms and smaller median project size; nevertheless, improvements remain statistically consistent across its repositories.

Open-set behavior is preserved at the language level: selective prediction (energy + entropy + variance) maintains closed-set accuracy while increasing TNR@TPR on withheld-class tests, enabling reliable confidence thresholds that map cleanly to CI policies (auto-action vs. human review).

Overall, Fig. 1, Fig.2 and Table 2 corroborate the central claim: multi-modal, hierarchical fusion delivers superior accuracy that transfers across languages without sacrificing operational safety.

Practical value. The proposed model yields immediate engineering benefits. It integrates into CI/CD as a confidence-calibrated gate, automating high-certainty cases while selective prediction escalates uncertain findings to reviewers. Incremental processing preserves low latency for pull requests, and hierarchical explanations (attention maps, channel weights) make alerts actionable and teachable during code review. Cross-language transferability and per-project normalization reduce retuning across repositories. In effect, the tool improves refactoring prioritization, cuts alert noise, stabilizes triage policies, and measurably elevates developer productivity.

Conclusions. The proposed hybrid, multi-level model-fusing CPG structure (AST+CFG+PDG), code semantics, classical metrics, and evolutionary signals with calibrated open-set selectivity – consistently outperforms single-view baselines. Averaged over repositories, Macro-AUPRC improves by +6.6 pp (Java), +7.3 pp (Kotlin), +6.7 pp (Scala) over the strongest baseline (AST+Text), while Macro-F1 gains are +6.5 pp, +7.3 pp, +6.9 pp, respectively (see Figure 8.5-8.6 and Table 8.3a,b). These gains come without increasing FPR@95TPR; probability calibration reduces ECE by approximately 25-30 % on validation, yielding predictable CI/CD thresholds.

Limitations. Labels rely on consensus detectors with limited human auditing; CPG/PDG completeness varies across languages and tools; framework/idiom drift can erode stability; evaluation focused on the JVM ecosystem, which may constrain external validity; deployment latencies can fluctuate with repository scale and CI policies.

Future Work. Move from detection to refactoring recommendation via counterfactual patches and test-verified edits; develop event- and causality-aware models of smell evolution; expand polyglot coverage (TypeScript/Go/Rust) and add modalities (tests/build telemetry); incorporate active learning for economical re-labeling; pursue online calibration and long-horizon threshold stabilization; study privacy/security implications of feature logging and explanation artifacts in production.

REFERENCES

1. Chidamber S. R., Kemerer C. F. “A Metrics Suite for Object Oriented Design”. *IEEE Transactions on Software Engineering.* 1994; 20 (6): 476–493. DOI: https://doi.org/10.1109/32.295895.

2. Ferrante J., Ottenstein K. J., Warren J. D. “The Program Dependence Graph and Its Use in Optimization”. *ACM Transactions on Programming Languages and Systems*. 1987; 9 (3): 319–349. DOI: https://doi.org/10.1145/24039.24041.

3. Palomba F., Bavota G., Di Penta M., Oliveto R., Poshyvanyk D. “Detecting Bad Smells in Source Code Using Change History”. *Proceedings of ASE*. 2013. p. 268–278. DOI: https://doi.org/10.1109/ASE.2013.6693086.

4. Azeem M. I., Palomba F., Shi L., Wang Q. “Machine Learning Techniques for Code Smell Detection: A Systematic Literature Review and Meta-Analysis”. *Information and Software Technology*. 2019; 108: 115–138. DOI: https://doi.org/10.1016/j.infsof.2018.12.009.

5. Wu Z., Pan S., Chen F., Long G., Zhang C., Yu P. S. “A Comprehensive Survey on Graph Neural Networks”. *IEEE Transactions on Neural Networks and Learning Systems*. 2021; 32 (1): 4–24. DOI: https://doi.org/10.1109/TNNLS.2020.2978386.

6. Veličković P., Cucurull G., Casanova A., Romero A., Liò P, Bengio Y. “Graph Attention Networks”. *arXiv*. 2017. DOI: https://doi.org/10.48550/arXiv.1710.10903.

7. Scheirer W. J., de Rezende Rocha A., Sapkota A., Boult T. E. “Toward Open Set Recognition”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2013; 35 (7): 1757–1772. DOI: https://doi.org/10.1109/TPAMI.2012.256.

8. Bendale A., Boult T. E. “Towards Open Set Deep Networks”. *Proceedings of CVPR*. 2016. p. 1563–1572. DOI: https://doi.org/10.1109/CVPR.2016.173.

DOI: https://doi.org/10.15276/ict.02.2025.45
УДК 004.8:004.415.2:519.17

Мультимодальні графові подання для надійного виявлення антипатернів в еволюційних кодових базах

Курінько Данило Дмитрович[1)]
Аспірант каф. Штучного інтелекту та аналізу даних
ORCID: https://orcid.org/0000-0001-8304-3257; dmitrykurinko@gmail.com
Кривда Вікторія Ігорівна[1)]
Канд. техніч. наук, доцент каф. Електропостачання та енергетичного менеджменту
ORCID: https://orcid.org/0000-0002-0930-1163; kryvda@op.edu.ua
[1)] Національний університет «Одеська політехніка», пр. Шевченка, 1. Одеса, 65044, Україна

АНОТАЦІЯ

У дослідженні оцінюється, чи підвищують мультимодальні та багаторівневі подання надійність виявлення запахів коду і антипатернів в еволюційних, мультимовних програмних системах. Запропоновано гібридну модель, що інтегрує чотири канали даних – структурний, семантичний, метричний та еволюційний – у єдиний Code Property Graph (CPG), який поєднує зв’язки AST, CFG і PDG. Семантичні відомості отримуються за допомогою попередньо навчених мовних моделей коду; класичні індикатори якості (CK, McCabe/Halstead) фіксуються як атрибути вузлів і ребер; сигнали систем контролю версій (churn, ко-зміни, давність) агрегуються з часовим згасанням для урахування актуальності. Навчання здійснюється ієрархічно: локальний енкодер підсумовує ідіоми на рівні токену та індуковані графові зрізи; компонентний, зв’язко-орієнтований GNN моделює когезію/зв’язування і структуру потоків даних/керування; проєктний енкодер поширює контекст у графі взаємодії компонентів. Екземплярно-залежний «гейтінг» каналів використовується для зважування модальностей і підкреслення релевантних ознак.

Для розгортання у відкритих умовах застосовано селективне передбачення з використанням взаємодоповнювальних критеріїв невизначеності (енергія логітів, ентропія, стохастична дисперсія) та температурне калібрування для покращення достовірності ймовірностей і можливості утримання від рішення у випадках низької впевненості. Емпірична оцінка охоплює репозиторії Java, Kotlin і Scala з міжпроєктними та часовими розбиттями; open-set тести формуються шляхом утримання класу смелів під час навчання. Порівняно з правилами/метриками, AST-GNN, текст-орієнтованими та AST+Text підходами, гібридна модель демонструє стабільні покращення без збільшення FPR@95TPR. У середньому по репозиторіях Macro-AUPRC зростає приблизно на 6-7 в. п., Macro-F1 – на 3-4 в. п., з найбільшими виграшами для God Class і Shotgun-Surgery-подібних категорій. Інкрементальні оновлення CPG і обмежена глибина пропагації забезпечують латентність, сумісну з CI/CD, а ієрархічні пояснення та ваги каналів надають інтерпретованість. Результати свідчать про багатосигнальну, контекстну природу запахів і ефективність ієрархічного, каліброваного, open-set підходу.

Ключові слова: машинне навчання; програмна інженерія; аналіз програм; графові моделі; статичний аналіз; виявлення антипатернів; якість програмного забезпечення; відкриті множини