Бублик В.В., Лозинський І.І., Серняк А.Ю.
Національний університет «Києво-Могилянська академія»

ДО ПИТАННЯ СТВОРЕННЯ СТРУКТУРНИХ МОДЕЛЕЙ ПРОГРАМНИХ СИСТЕМ

Створення і дослідження моделей програм почалося практично одночасно з виникненням програмування в середині минулого сторіччя. Створені моделі охоплювали всі аспекти програмування: синтаксичний, семантичний і прагматичний. Повний перелік напрямків і підходів звісно виходить за рамки цього повідомлення, але деякі з них, зокрема ті, що справили особливий вплив на роботу авторів варто згадати. На рівні синтаксичному успішно розвивалися моделі, що спиралися на теорію формальних мов і граматик, теорію автоматів та схем програм (А.Н.Хомський, Л.А.Калужнін, В.М.Глушков, Ю.І.Янов, О.А.Летичевський) [1]. На семантичному рівні автоматні методи операційної семантики доповнювалися методами новостворених програмних логік та математичної теорії обчислень (віденський метод, роботи К.Стретчі та Д.Скота, Р.Флойда, Е.Дейкстри, Ч.А.Р.Хоора, А.П.Єршова, В.Н.Редька) [2,3]. Прагматичний рівень доповнив два попередніх цілим спектром діаграм, що торкалися широкого кола питань, стосовних різних етапів життєвого циклу як окремих програм, так і цілих програмних систем. Досить згадати блок-схеми програм, що протягом тривалого часу були необхідним компонентом програмної документації. *Сюди ж можна віднести методи графічного програмування*, зокрема, Р-технологію І.В.Вельбицького і в першу чергу діаграми UML [4], що охопили практично всі етапи життєвого циклу, починаючи з аналізу предметної області і закінчуючи розгортанням, супроводом і рефакторингом програмних систем.

Згадані вище методи моделювання програмних систем орієнтовані перш за все на процеси їх розроблення. Дійсно, програмна система, розроблена, спроектована і втілена на основі певної моделі служить варіантом її реалізації тією мірою, якою супровід, що включає в себе продовження розроблення, і, особливо, рефакторинг впливають на стан і склад моделі. Як правило, моделі програмних систем статичні, а внесення змін до програмної системи в процесі її супроводу надає системі динамізму, все більше віддаляючи її від висхідної моделі. В даній роботі автори знаходяться на позиції пошуку методів відновлення моделі, виходячи з наявного стану програмної системи. Відновлена в такий спосіб модель допоможе колективу розробки і супроводу ефективно супроводжувати програмну систему на подальших етапах її життєвого шляху.

Підхід авторів певною мірою визначено впливом фундаментальних концепцій моделювання [5], що набувають особливого значення у випадку моделювання великих і надвеликих програмних систем. Паралельно зі зростанням функціональних і операційних вимог складність таких систем стає критичною.

Особливість фундаментальної концепції полягає у поєднанні технічних аспектів моделей, покликаних сприяти створенню системи, з комунікаційними, призначеними для досягнення ефективного взаєморозуміння членів команди розробників і супроводу.

Ефективні методи для обміну знаннями про програмно керовану систему серед учасників процесу її створення та експлуатації, розподілених як в просторі, так і в часі, з ростом її складності набувають вирішального значення. Всі учасники повинні мати загальне уявлення про систему в цілому, з тим щоб ефективно оперувати проектними рішеннями, оцінювати, розвивати та реалізовувати їх. Визначення та візуалізації структури всієї системи виявляться корисними в цьому процесі. У будь-якому випадку необхідно поєднувати три складові частини: термінологічну, відповідальну за точність вживаних понять і конструкцій; структурну, що визначатиме модель для відтворення структури системи; нотаційну, призначену для сприяння людському розумінню.

Першим кроком в досягненні широкомасштабного моделювання програмно керованих систем автори пропонують використання спеціалізованої утиліти «Monte». Підхід авторів до створення структурної моделі програмних систем мотивується прагматичними вимогами, що викристалізувалися в процесі роботи над утилітою, зокрема в рамках обговорення групових проектів в курсі «Методи програмної інженерії», що читається на магістерській програмі факультету інформатики Національного університету «Києво-Могилянська академія». До цих вимог відносяться: простота у застосуваннях; орієнтація на розробника, а не технології; спрямованість на повторні застосування.

Процес створення моделі складається з трьох етапів. На першому відбувається накопичення даних. В «Monte» для цього використовуються «майнери» (від англійського «miner»): невеликі плагіни, що самі аналізують код чи інші проектні артефакти або ж використовують результати роботи зовнішніх утиліт. У розробника є можливість реалізувати свій «майнер», щоб інтегрувати вже існуючі рішення, адже іноді доцільніше й простіше повторно використати наявний код, особливо зважаючи на те, що певні мови програмування за замовчуванням надають розробнику інструменти для динамічного аналізу програмного коду, такі як інтроспекція чи рефлекція. Також запропонований підхід дозволяє використати дані, що зазвичай ігноруються при аналізі, але які можуть містити корисну інформацію щодо структури програмної системи, як-от структура директорій в проекті, назви файлів, назви та специфікації програмних взірців тощо.

На другому етапі відбувається аналіз накопичених даних, що полягає в наданні їм конкретних семантичних властивостей та зв'язків. Крім цього дані уніфікуються (приводяться до єдиного вигляду) й у них усувається надлишковість. Також на цьому етапі додаються метрики, корисні для подальшого аналізу й візуалізації. Для прикладу наявні «майнери» повертають список імен розробників, назв класів, завдань та історію фіксацій з системи контролю версій. Якщо в зібраних наборах даних деякі сутності зустрічаються в різних форматах,

то проводяться уніфікація та групування. Далі конкретного розробника можна пов'язати з класами через історію фіксацій, класи – із завданнями через повідомлення фіксацій тощо. Також можна додати метрики, такі як загальна кількість чи частота фіксацій, зроблених розробником. Результатом цього етапу є узагальнена структурна модель програмного проекту.

На третьому етапі узагальнена структурна модель візуалізується з різною мірою деталізації за допомогою переглядів, визначених користувачем. До переглядів застосовні групування, фільтрація та пошук. Одночасно можна оперувати кількома переглядами. Набір таких переглядів слугуватиме «структурною картою» для команди розробників і супроводу. Для попереднього прикладу можна визначити перегляди для графічного зображення класів, розробників, завдань та зв'язків між ними. Усі дані прив'язані до часових інтервалів, що забезпечує динамічність розгортання моделі. Також можна задати перегляд для тестів, що міститиме тестові сценарії, результати виконання тестів тощо. Крім того запропонований підхід дозволяє створювати додаткові перегляди в залежності від потреб команди розробників і супроводу.

Наразі авторами проводяться експериментальні дослідження можливостей запропонованого підходу.

Література:

1. Глушков В.М. Алгебра. Языки. Программирование / Глушков В.М., Цейтлин Г.Е., Ющенко Е.Л., Киев. Наукова думка. – 1989.– 317 с.

2. Грис Д. Наука программирования / Грис Д. — М. Мир , 1984. — 416 с.

3. Ершов А. П. Введение в теоретическое программирование / Ершов А. П. — М. Наука, 1977. — 288 с.

4. Буч Г. UML. Классика CS. / Буч Г, Якобсон А., Рамбо Дж.– Питер. СПб., 2006. – 736 с.

5. Andreas Knöpfel. Fundamental Modeling Concepts: Effective Communication of IT Systems / Andreas Knöpfel, Bernhard Gröne, Peter Tabeling. — Wiley, 2006, — 350 pp.