Тема: Проблема надійності програмного забезпечення

Восени 1968 і 1969 років НАТО організувало конференції, присвячені темі розробки програмного забезпечення, хоча цей термін не був загальновживаним у той час. Причиною скликання цих конференцій стало те, що (тогочасна) комп'ютерна індустрія мала великі труднощі у створенні великих і складних програмних систем. (Ця труднощі й досі нікуди не поділася, хоча й дещо змінилися, а проблеми, які вони викликають тільки загострилися). До участі в цих конференціях запрошували виробників та користувачів комп'ютерів, виробників програмного забезпечення та ведучих науковців. В певному розумінні можна стверджувати, що ці конференції сфокусували увагу на проблемі, що таке ``надійність'' програмної системи і як її забезпечити в процесі розробки.
Цей курс ми почнемо з обговорення складного і, навіть, суперечливого поняття - поняття надійності для програмної системи. Мета нашого обговорення - визначити критичну, з точки зору надійності, проблему процесу розробки програмних систем.
Почнемо зі спроби відповісти на питання, чому труднощі розробки програмних систем, які обговорювалися у 1968-1969 роках, і сьогодні залишилися неподоланими. Більш того на цей час, як вже зазначено, вони навіть загострилися зважаючи на тотальне використання складних програмних систем.

1. Системи з інтенсивним використанням програмного забезпечення

На нашу думку, джерелом цих труднощів є тренд розвитку сучасної техніки -- від розділення технічних засобів на ті, що призначені для трансформації суто матеріальних (енергетичних) потоків, і на ті, що призначених для трансформації суто інформаційних потоків, до засобів, які поєднують ці два види потоків у єдине ціле, з метою досягнення синергетичного ефекту.
Наслідком цього тренду стало виникнення цілих класів технічних систем таких, як вбудовані системи (an embedded system), кібер-фізичні системи (a cyber-physical system), інтернет речей (Internet of Things) тощо. Загалом технічні системи цих класів можна охарактеризувати як системи з інтенсивним використанням програмного забезпечення (a software intensive system).
Тобто зазначений вище тренд веде до домінування класу системи з інтенсивним використанням програмного забезпечення серед технічних систем в цілому.
Що ж мають на увазі, коли використовують термін система з інтенсивним використанням програмного забезпечення?
Стандарт IEEE-Std-1471-2000 визначає систему з інтенсивним використанням програмного забезпечення, як будь-яку систему, у якій програмне забезпечення істотно впливає на проектування, створення, розгортання, функціонування та подальший розвиток системи в цілому.
Зростання важливості цього класу систем ілюструється наступною таблицею, що використовує дані компанії IDate, яка дає частку програмного забезпечення у доданій вартості продукції різних галузей індустрії.

Таблиця 1: Частка програмного забезпечення у доданій вартості продукції різних галузей індустрії

Галузь промисловості	**Частка ПЗ у доданій вартості**	
	2002 рік	**2015 рік**
Автомобільна промисловість	3,37%	6,41%
Аерокосмічна прмисловість	9,38%	12,06%
Побутова електроніка	7,98%	11,40%
Виробництво медичного обладнання	6,25%	9,90%
Автоматизація	1,00%	1,50%
Виробництво телекомунікаційного обладнання	13,00%	16,25%

Для системи з інтенсивним використанням програмного забезпечення коректне функціонування програмних компонентів такої системи є критичним для забезпечення коректного функціонування системи в цілому. Наслідками некоректного функціонування програмних компонентів є впливи на системне оточення, які можна характеризувати як техногенні катастрофи.
Ось низка відомих інцендентів.

1.1. Космічний глюк

4 червня 1996 року Європейське космічне агентство запустило ракету-носій сімейства Аріан, призначену для виведення корисного навантаження на низьку опорну орбіту. Ракета, що стартувала з космодрому Куру у Французькій Гвіані, через 37 секунд після зльоту вибухнула у повітрі, засіявши околиці уламками.

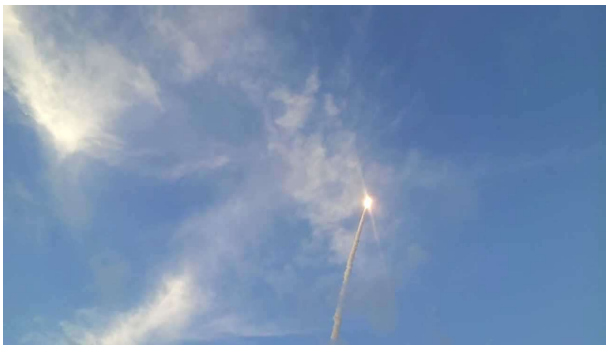

Рис. 1: Загибель Аріан-5

На щастя, ніхто не постраждав. Фахівці підозрювали витік палива чи проблеми з електронікою. Але все виявилося набагато прозаїчнішим: помилка в програмному забезпеченні модуля управління призвела до самознищення ракети. Мільйони євро згоріли в повітрі через помилку в коді.

1.2. Збій вартістю мільярд доларів

2013 року капіталізація великої інвестиційної компанії Knight Capital становила кілька мільярдів доларів. Вона цілком справедливо вважалася однією з найуспішніших інвестиційних груп у США. Проте всього за півгодини її фінансовий успіх перетворився на спогад: збій комп'ютерної програми повністю розорив компанію.

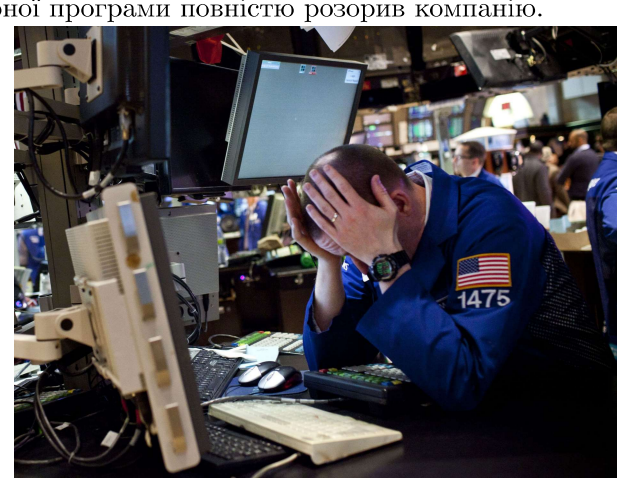

Рис. 2: Банкрутство Knight Capital

Комп'ютери почали неконтрольовано купувати та продавати мільйони акцій, допоки трейдери безпорадно стукали по клавішах. За феєрично короткий час Knight Capital втратила півмільярда доларів. Підсумком збою стало падіння акцій компанії на 75% - всього за 2 дні перспективна фірма практично повністю розорилася.

1.3. Помилка у програмному забезпеченні медичного призначення

У 80-ті роки минулого століття мінімум 6 пацієнтів один за одним померли від отримання великої дози рентгенівського випромінювання під час лікування з використанням апарату для променевої терапії Therac-25, що застосовується в процесі лікування раку.

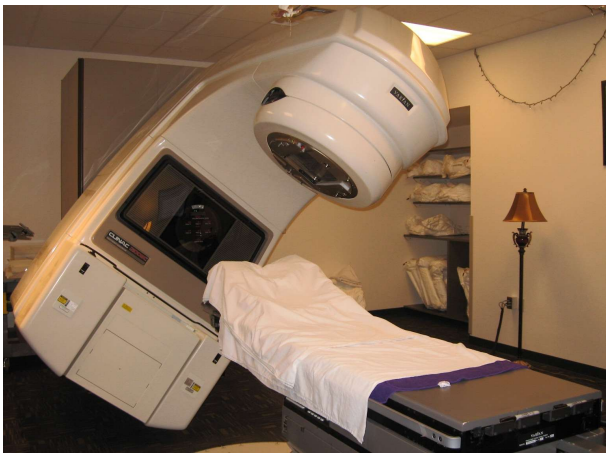

Рис. 3: Інцендент з Therac-25

Деякі пацієнти отримали дози в десятки тисяч рад – і все через програмну помилку блоку управління установки. Цікаво, що проблему не помічали протягом кількох років. Експерти, які вивчили систему, встановили, що збій був викликаний помилкою в коді, в результаті якої програма намагалася виконувати ту саму команду багаторазово. Замість опромінення у медичних дозах пацієнти помирали від передозування радіацією. Ця програмна помилка вважається однією з найгірших наслідків з вини програмного забезпечення за історію використання комп'ютерів.

1.4. Гроза у торговій мережі

У серпні 2014 року протягом 45 хвилин сервери Amazon – загальновідомого найбільшого у світі сайту серед тих, хто продає товари та послуги через Інтернет – були недоступні через відключення електрики.

Рис. 4: Гроза проти Amazon

Спочатку аварія, викликана сильною грозою, посилилася численними ускладненнями у вигляді програмних помилок, що раптово спливли, в результаті чого стався каскадний збій. За час простою сайт втратив близько $5 млн. На щастя, сайт більш ніж серйозно підходить до безпеки даних, тому рядові користувачі не втратили ні грошей, ні вже оплачених, але ще не доставлених замовлень.

1.5. Електронна пітьма

До чого може призвести маленька, непомітна помилка в програмному забезпеченні для моніторингу роботи обладнання General Electric Energy? 2003 року приблизно третина штатів США залишилася без електроенергії взагалі.

Рис. 5: Блекаут на північному сході США

Світла не було не лише у будинках громадян, а й у лікарнях, школах, на вокзалах. Не працювали комп'ютери у поліцейських дільницях, не функціонували радари сил протиповітряної оборони, літаки не могли піднятися зі злітних смуг. Гірше того: замовкли диспетчерські вежі, і кілька страшних годин літаки, що знаходяться в повітрі, не могли сісти. Глобальний каскадний збій, спричинений непомітною помилкою, спричинив одне з наймасштабніших відключень електроенергії в історії.

1.6. Масове приземлення

Через виявлену в 2013 році помилку у програмному забезпеченні систем контролю польотів керівництву авіаперевізника American Airlines довелося повернути на землю абсолютно всі свої літаки.

Рис. 6: American Airlines приземлила всі свої літаки

Збій виник у системі бронювання квитків, коли внаслідок злиття кількох авіакомпаній дві існуючі системи об'єдналися в одну. Проблема з керуванням польотами виникла після того, як спроба об'єднати платформи, написані різними мовами програмування, провалилася.
Підсумовуючи, можемо зазначити наявність **протиріччя** між низькою інтенсивністю використання програмного забезпечення у галузі автоматизації та катастрофічно високою ціною помилок розробники автоматизованих систем.

2. Концепція надійності (dependability)

Велика українська енциклопедія визначає надійність, як властивість технічних об'єктів зберігати у часі в установлених межах значення всіх параметрів, які характеризують здатність виконувати потрібні функції в заданих режимах та умовах застосування, технічного обслуговування (ТО), зберігання та транспортування.
Стандарт ДСТУ 2860-94 визначає надійність, як властивість технічних об'єктів зберігати протягом встановленого часу значення всіх параметрів, які характеризують здатність виконувати потрібні функції в заданих режимах та умовах застосування, технічного обслуговування, зберігання та транспортування.
Тобто, обидва джерела дають майєже ідентичні визначення надійності технічних\linebreak об'єктів.
Однак аналіз цих визначень дозволяє встановити, що надійність розглядається не як властивість притаманна технічному об'єкту безвідносно до часу його функціонування, а як властивість, що притаманна технічному об'єкту тільки протягом певного інтервала часу.
Причиною такого обмеження є об'єктивні процеси фізичного старіння матеріальних об'єктів, що призводять до зміни властивостей цих об'єктів з плином часу.
Натомість, програми, як об'єкти інформаційні, а не матеріальні, не підлягають впливам процесів фізичного старіння. Зважаючи на це, часові обмеження періоду функціонування втрачають сенс.
Слід зазначити, що англомовні джерела використовують два терміни "reliability" та "dependability", які обидва перекладаються українською як надійність.
International Electrotechnical Commission (IEC) визначє надійність (reliability), як "ability to

perform as required, without failure, for a given time interval, under given conditions".
Тобто термін надійність (reliability) визначається так само, як визначає термін надійність стандарт ДСТУ 2860-94.
Натомість, IEC визначає термін надійність (dependability), як "ability to perform as and when required".
Тобто термін надійність (reliability) має ймовірностний характер і оцінюється зазвичай, як вірогідність коректного функціонування системи протягом певного інтервалу часу, а термін надійність (dependability), натомість, має якісний характер і розділяє системи на ті, функціонування яких є коректним (гідні довіри), і ті, функціонування яких є некоректним.
Тому IEC вважає, що термін надійність (dependability) посилається на комплексну характеристику системи.
Ця характеристика може бути описана моделлю надійності (dependability), яка є концептуальною основою, що використовується для оцінки та забезпечення надійності, доступності, безпеки та інших відповідних атрибутів системи.

2.1. Модель надійності

Ключовими компонентами моделі надійності є:

- ***надійність (reliability)*** оцінює ймовірність того, що система виконуватиме заплановану функцію без збоїв протягом певного періоду часу. Його часто вимірюють за допомогою середнього часу напрацювання на відмову (MTBF) або частоти відмов;
- ***доступність (availability)*** вимірює частку часу, протягом якого система працює та доступна. Включає такі фактори, як планове технічне обслуговування, простої через збої та час відновлення;
- ***безпека (safety)*** - цей аспект безпеки зосереджений на запобіганні нещасним випадкам, травмам і втраті життя чи власності. Це включає визначення потенційних небезпек, аналіз ризиків і впровадження заходів для їх зменшення;
- ***безпека (security)*** - цей аспект безпеки зосереджений на захисті від несанкціонованого доступу, атак і витоку даних. Це включає такі заходи, як шифрування, контроль доступу та системи виявлення вторгнень;
- ***ремонтопридатність (maintainability)*** оцінює, наскільки легко систему можна відремонтувати, оновити або модифікувати. Враховуються такі фактори, як середній час до ремонту (MTTR) і легкість діагностики проблем;
- ***надмірність (redundancy)*** означає наявність резервних компонентів або систем, які можуть взяти на себе роботу в разі збою, підвищуючи надійність (reliability) системи;
- ***відмовостійкість (fault tolerance)*** є здатністю системи продовжувати належну роботу навіть за наявності несправностей або збоїв;
- ***тестування та валідація (testing and validation)*** передбачає низку випробувань та перевірок, щоб переконатися, що продуктивність системи відповідає визначеним вимогам до надійності.

Модель надійності може також включати фактори, пов'язані з середовищем, в якому працює система, складністю системи, критичністю її функцій і потенційними наслідками відмов. Моделі надійності можуть бути формальними, математичними представленнями або більш якісними рекомендаціями, залежно від складності системи та бажаного рівня довіри. Вони відіграють вирішальну роль у різних галузях, включаючи аерокосмічну галузь, охорону здоров'я, промислову автоматизацію та критичну інфраструктуру, де надійність і

безпека мають першочергове значення.

2.2. Загрози руйнування надійності

Надійність (dependability) може бути зруйнована загрозами, які класифікуються, як

- ***відмова (failure)*** - подія, що полягає у втраті об'єктом здатності виконувати потрібну функцію, тобто у порушенні працездатного стану системи;
- ***збій (fault)*** - самоусувна відмова або одноразова відмова, що усувається незначним зовнішнім втручанням;
- ***помилка (error)*** - невідповідність між запланованою поведінкою системи та її фактичною поведінкою, що виникає під час виконання, коли якась частина системи переходить у неочікуваний стан через відмову.

Інакше кажучи, помилка є причиною відмов та збоїв і реалізується через них.
Проте система може мати приховані помилки, тобто помилки, які з огляду на фактичні умови функціонування системи не реалізуються через відмови.
Підсумовуючи, можна зробити наступні висновки.

1. Для програмного забезпечення критичною загрозою є помилка в програмному коді, яка може викликати напрогнозовану поведенку системи в цілому.
2. Помилку в програмному коді можна визначити як розбіжність між фактичними властивостями поведінки розробленого програмного коду і заявленими у специфікації вимог властивостями поведінки програмного коду.
3. Забезпечення надійності (dependability), таким чином, передбачає комплекс захдів щодо встановлення істинності судження ``програмний код відповідає властивостям, що зафіксовані у специфікації вимог''.

3. Динамічний і статичний аналіз коду. Сертифіковане програмування

Методи, на яких базуються зазначені вище заходи щодо встановлення істинності судження ``програмний код відповідає властивостям, що зафіксовані у специфікації вимог'', реалізують два принципово різних підходи - динамічний і статичний аналіз коду.

3.1. Динамічний аналіз коду

Методи динамічного аналіза коду є достатньо традіційними.
Всі вони базуються на безпосередніх спостереженнях за виконанням коду.
Проблема, однак, полягає в тому, що самі інструменти спостереження за виконанням коду можуть суттєво і непередбачувано впливати на таке виконання, спотворюючи результати спостереження, як це буває, наприклад, для розподілених програмних систем.
Крім того, методи динамічного аналіза хоча і дозволяють виявити певні помилки, проте вони не дають гарантії відсутності помилок у разі невиявлення останіх.

3.2. Статичний аналіз коду

Натомість, методи статичного аналіза коду не передбачають виконання коду.
Вони відомі як методи формальної верифікації.
Ідея підходу полягає в тому, що паралельно з розробкою коду має будуватися його сертифікат - формальне доведення властивостей, що визначені специфікаціями вимог.
Саме з огляду на це, підход називають сертифікованим програмуванням.
Цей підхід пов'язаний з двома проблемами

- об'єктивною, що є наслідком теоремои Райса, яка заперечує розв'язність будь-якої

нетривіальної семантичної властивості програм у рамках повних за Тьюрингом систем програмування;

- суб'єктивною (технічною), яка пов'язана з високим математичним рівнем входження в проблематику, що суттєво звужує коло фахівців, які можуть використовувати цей підхід.

3.3. Мета курсу

Метою нашого курсу якраз і є ознайомлення слухачів з сертифікованим програмуванням, базуясь на використанні комплексу програмних інструментів, відомих як [The Coq Proof Assistant](https://coq.inria.fr/).
The Coq Proof Assistant є інструментальною системою підтримки формальних доведень.
Вона включає формальну мову для специфікації математичних визначень, алгоритмів і тверджень разом із середовищем для інтерактивної розробки доведень, перевірених програмним забезпеченням системи.
Типові застосування включають

- сертифікацію властивостей мов програмування (наприклад, проект сертифікації компілятора CompCert, Verified Software Toolchain для перевірки програм C або фреймворк Iris для логіки паралельного розділення),
- формалізацію математики (наприклад, повну формалізацію теореми Фієта-Томпсона, або теорії гомотопічних типів).

Вона також використовується у навчальних цілях.