7.Техники (тактики) за постигане на качеството на софтуера (част 2)-Резюме

1.Тактики за изменяемост (Modifiability)

-Локализиране на промените: Намаляване на броя на модулите, засегнати от промяна.

Семантична свързаност (Semantic Coherence): Обединяване на функционалности, които са семантично свързани в един модул.

Очакване на промените: Списък на най-вероятните промени и оценка на декомпозицията. Помага ли така направената декомпозиция да бъдат локализирани необходимите модификации за постигане на промяната? Случва ли се така, че фундаментално различни промени да засягат един и същ модул?

Ограничаване на възможните опции: Намаляване на вариациите, които могат да засегнат много модули.

-Предотвратяване на ефекта на вълната: Ограничаване на модификациите до директно засегнатите модули. Ефект на вълната има тогава, когато се налагат модификации в модули, които не са директно засегнати от дадена промяна.

Скриване на информация (Information Hiding): – декомпозиция на отговорността на даден елемент (система или конкретен модул) и възлагането ѝ на по-малки елементи, като при това част от информацията остава публична и част от нея се скрива. Публичната функционалност и данни са достъпни посредством специално дефинирани за целта интерфейси. Това е най-старата и изпитана техника за **ограничаване на промените** и е пряко свързана с “**очакване на промените**”, тъй като именно списъка с очакваните промени е водещ при съставянето на декомпозицията, така че промените да бъдат сведени в рамките на отделни модули.

Ограничаване на комуникацията, Скриване на информация, Facade: Намаляване на броя на модулите, с които даден модул обменя информация - ограничават се модулите, които консумират данни, създадени от модул А и се ограничават модулите, които създават информация, която се използва от модул А.

Поддръжка на съществуващите интерфейси: Ако Б зависи от даден интерфейс от А, то съответният синтаксис трябва да се поддържа непроменен. За целта се прилагат следните техники:

-Добавяне на нов интерфейс – вместо да се сменя интерфейс се добавя нов;

-Добавя се адаптер – А се променя и същевременно се добавя адаптер, чрез който се експлоатира стария синтаксис;

-Създава се stub – ако се налага А да се премахне, на негово място се оставя stub – процедура със същия синтаксис, която обаче не прави нищо (NOOP);

Използване на посредник (Mediator): Ако Б зависи по някакъв начин от А (освен семантично), е възможно между А и Б да бъде поставен „посредник“, които премахва тази зависимост: wrapper, facade, adaptor, proxy, mediator

-Отлагане на свързването: Контролиране на времето за внедряване и себестойността на промяната.

Plug-and-Play: Включване/изкл./замяна на компоненти, както по време на изпълнението (plug-and-play), така и по време на зареждането (component replacement)

Конфигурационни файлове: Задаване на стойности на параметри чрез конфигурационни файлове.

Протоколи: Дефиниране на протоколи, които позволяват промяна на компоненти по време на изпълнение.

2. Тактики за сигурност (Security)

- **Устояване на атаки (ключалка на вратата)**:

Authentication/Автентикация/ на потребителите – проверка за това, дали потребителя е този, за който се представя (пароли, сертификати, биометрика и т.н.)
Authorization /Оторизация/ на потребителите – проверка за това дали потребителя има достъп до определени ресурси

Confidentiality/Конфиденциалност/ на данните – посредством криптиране (на комуникационните канали и на постоянната памет)

Integrity/Интегритет/- включване на различни механизми на излишък – чек-суми, хеш-алгоритми и т.н.

Ограничаване на експозицията - местата, чрез които можем да бъдем атакувани

Ограничаване на достъпа- firewall и др. средства за ограничаване на физическия достъп.

- **Откриване на атаки (аларма)**:

Monitoring: Наблюдение на системата за подозрителна активност.

Intrusion Detection Systems (IDS): Системи за откриване на проникванията.

- **Възстановяване след атака (застраховка) -свързани с възстановяване на състоянието и с идентификация на извършителя.**

Тактиките за възстановяване на системата донякъде се препокриват с тактиките за Изправност, тъй като извършена атака може да се разгледа като друг срив в работата на системата. Трябва да се внимава обаче в детайли като пароли, списъци за достъп, потребителски профили и т.н.

Тактиката за идентифициране на атакуващия-Audit Trail: Поддръжка на копие на всяка транзакция за идентификация на извършителя и възстановяване от атаката. Audit trail-овете също са обект на атака, така че при проектирането на С трябва да се вземат мерки достъп до тях да се осигурява само при определени условия.

3. Тактики за удобство при тестването или Тактики за изпитаемост (Testability) - Целта на тактиките за изпитаемост е да подпомогнат тестването, когато част от софтуерната разработка е приключила. За фазата преди приключването на разработката за това обикновено се прилагат тактики като code review. За тестване на работеща система обикновено се използва софтуер, който чрез изпълнението на специализирани скриптове подава входни данни на системата и анализира резултата от нейната работа. Целта на тактиките за изпитаемост е да подпомагат този процес

Специализиран интерфейс за тестване: Предоставяне на интерфейс за тестващ софтуер, различен от нормалния.

Вградени модули за мониторинг: Самата система поддържа информация за състоянието, натовареността, производителността, сигурността и т.н. и я предоставя на определен интерфейс. Интерфейсът може да е постоянен или временен, включен посредством техника за инструментиране.

4. Тактики за удобство при употребата (Usability) - Използваемостта се занимава с това, колко лесно даден потребител успява да свърши дадена задача и доколко системата подпомага това му действие. Различаваме два вида тактики за използваемост, всяка от тях насочена към различна категория “потребители”:

-Тактики за използваемост **по време на изпълнението** (runtime) – **насочени към крайния потребител**

-Тактики за използваемост, **свързани с UI** – **насочени към разработчика на интерфейса**. Този вид тактики са силно свързани с тактиките за изменяемост

- **По време на изпълнението**: По време на работа на системата, използваемостта обикновено е свързана с предоставянето на достатъчно информация, обратна връзка и възможност за изпълнение на конкретни команди (cancel, undo, aggregate, multiple views и т.н.) В областта на Human-Computer Interaction (HCI) обикновено се говори за "**потребителска инициатива**", "**системна инициатива**" или "**смесена инициатива**". Напр., когато се отказва дадена команда, потребителя извършва "cancel" (потребителска инициатива) и системата отговаря. По време на извършването на това действие обаче, системата предоставя индикатор за прогреса (системна инициатива).

Обратна връзка: Предоставяне на достатъчно информация и възможност за изпълнение на команди като cancel, undo, multiple views.

Моделиране на задачата: **Системата** знае какво прави потребителят и му помага (напр. AutoCorrection).

Моделиране на потребителя: **Системата** се адаптира към поведението на потребителя.

Моделиране на системата: **Системата** предоставя адекватна обратна връзка (напр. progress bar).

- **Свързани с UI**: **Разделяне на интерфейса от реализацията**: Например при Model-View-Controller (MVC).

Заключение: Постигането на качествата на софтуерната система изисква както архитектурни, така и не-архитектурни решения. Качествата не могат да се разглеждат в изолация, тъй като постигането на едно качество може да повлияе на друго.