6.Техники (тактики) за постигане на качеството на софтуера

Архитектурни тактики: Фундаментални арх. решения, чрез които се контролират резултатите на даден сценарий за качество. Наборът от конкретни тактики се нарича архитектурна стратегия.

СА определя структурите на системата в контекста на нефункционалните изисквания. Интересът на софтуерния архитект към функционалността обикновено е ограничен до въпроса доколко тя взаимодейства с тези нефункционални изисквания (най-често ограничавайки ги).

Функционалността често е единственото, което се взема под внимание през проектирането. Като следствие много С се преправят не защото им липсва функционалност, а защото е трудно да се поддържат, трудно е да се смени платформата, не са скалируеми, прекалено са бавни, или пък са несигурни. СА е тази стъпка в процеса на създаването на С, в която за пръв път се разглеждат качествените изисквания и в зависимост от тях се създават съответните структури, на които се приписва функционалност.

1. Тактики за изправност (Dependability): Съвкупност от няколко качества на софт. С, вкл. **Availability**– Достъпност, **Reliability**– Надеждност, **Safety**–Безопасност, **Integrity**–Цялост, **Confidentiality**– Поверителност, **Maintainability**– Поддържане. Вкл.:

-Откриване на откази

1.Exo(Ping/Echo): Компонент А изпраща сигнал до компонент Б и очаква отговор в рамките на опр. интервал от време. Ако отговорът не се получи навреме, се предполага, че в компонент Б е настъпила повреда и се задейства процедурата за отстраняване на повредата.

2.Heartbeat/Keepalive: Компонент периодично излъчва сигнал, който друг компонент очаква. Ако сигналът не се получи, се предполага, че в компонент А е настъпила повреда и се задейства

процедура за отстраняване на повредата. Сигналът може да носи и полезни данни – пр: банкоматът може да изпраща журнала от последната транзакция на даден сървър. Сигналът не само действа като heartbeat, но и служи за лог-ване на извършените транзакции

3. Изключения(Exceptions): Обработка на изключения, генерирани при определено състояние на отказ.

-Отстраняване на откази

1. Активен излишък (Active Redundancy, Hot Restart): Дублиране на важни компоненти, които се поддържат в едно и също състояние. Използва се резултатът само от единния от компонентите (т.н. активен); Пр.:Използва в клиент/сървър конфигурация, където се налага бърз отговор дори при срив. Освен излишък в изчислителните звена се практикува и в комуникациите, в хардуера и т.н. Downtime-ът обикновено се свежда до няколко милисекунди, тъй като резервният компонент е готов за действие и единственото, което трябва да се направи е той да се направи активен.

2. Пасивен излишък (Passive redundancy, warm restart) – Основният компонент реагира на събитията и информира резервните за промяната на състоянието. При откриване на отказ, преди да се направи превключването на активния компонент, С трябва да се увери, че новият активен компонент е в достатъчно осъвременено състояние. Обикновено се практикува периодично форсиране на превключването с цел повишаване на надеждността. Downtime-ът е от няколко секунди до няколко часа. Синхронизацията се реализира от активния компонент

3. Резерва (Spare): Поддръжка на резервни изчислителни мощности, които се инициализират и пуснат в действие при отказ компонент. За целта е необходима постоянна памет, в която се записва състоянието на С и която може да се използва от резервната система за възстановяване на състоянието. Обикновено се използва за

хардуерни компоненти и работни станции. Downtime-ът обикновено е от няколко минути до няколко часа.

Основни предизвикателства: Синхронизация на състоянието на отделните дублирани модули; Данните трябва да са консистентни във всеки един момент

-Разнородност (Diversity)- Отказите в софтуера обикновено се предизвикват от грешки при проектирането. Мултиплицирането на грешка в проектирането чрез репликация не е добра идея. Просто увеличаване на броя идентични копия на програмата не е решение. Трябва да се въведе разнородност в копията на програмата:

1. **Разнородност в проектирането (design diversity)**: Използване на разл. програмни езици, компилатори, алгоритми, Ограничен/липса на контакт между отделните екипи. Техники:

Recovery Blocks- Разработват се няколко алтернативни модула на програмата

Програмиране на N на брой версии (N-version programming) - Voting - На различни процесори работят еквивалентни процеси, като всички те получават един и същ вход и генерират един и същ резултат, който се изпраща на т.н. **voter** (output comparator), който решава крайния резултат от изчислението. Ако някои от процесите произведе изход, който се различава значително от останалите, voter-ът решава да го изключи от обработката. Алгоритъмът за изключване на процес от обработката може да бъде различен и изменян, пр. отхвърляне чрез мнозинство, предпочитан резултат т.н

2. **Разнородност по данни (data diversity)**: Различ. данни за тестове.

3. **Разнородност по време (temporal diversity)**: Изпълнение на програмата в разл. моменти от време. Предполага възникването на определени събития, които касаят работата на програмата, по различно време. Методи за реализация на разнородност по време:

1/Чрез стартиране на изпълнението в различни моменти от време
2/Чрез подаване на данни, които се използват или четат в различни моменти от времето

Извеждане от употреба (Removal from service) – премахва се даден компонент от системата, за да се избегнат очаквани сривове. Пр. периодичен reboot на сървърите за да не се получават memory leaks и така да се стигне до срив. Извеждането от употреба може да става автоматично и ръчно, като и в двата случая това следва да е предвидено в С на ниво архитектура. Предполага наличието на модул за наблюдение (monitoring)

Следене на процесите (Process Monitoring) – посредством специален процес се следят основните процеси в системата. Ако даден процес откаже, мониторинг процеса може да го премахне, преинициализира, да създаде нов екземпляр и т.н.

-Повторно въвеждане в употреба

Паралелна работа (shadow mode) – преди да се въведе в употреба компонент, който е бил повреден, известно време се оставя той да работи в паралел в системата, за да се уверим, че се държи коректно, точно както работещите компоненти.

Контролни точки и rollback (Checkpoint/rollback) – **Контролната точка** е запис на консистентно състояние, създаван периодично или в резултат на определени събития. Понякога С се разваля по необичаен начин и изпада в не-консистентно състояние. В тези случаи, системата се **възстановява (rollback)** в последното **консистентно състояние (съгласно последната контролна точка) и журнала на транзакциите**, които са се случили след това.

2. Тактики за производителност (Performance) **Цел**: Постигане на реакция от страна на С на зададено събитие в рамките на определени времеви изисквания. За да реагира системата е нужно време, защото:
- Ресурсите, заети в обработката го консумират;

- Защото работата на C е блокирана поради съревнование за ресурсите, не-наличието на такива, или поради изчакване на друго изчисление;

Тактиките за производителност са разделени в 3 групи: **Намаляване на изискванията**; **Управление на ресурсите**; **Арбитраж на ресурсите**;

-Намаляване на изискванията

-Увеличаване на производителността на изчисленията: Подобряване на алгоритмите, замяна на един вид ресурси с друг (напр. кеширане)

-Намаляване на режийните (overhead): не-извършване на всякакви изчисления, които не са свързани конкретно с конкретното събитие

-Промяна на периода: Редуциране на честотата на периодични събития.

-Промяна на тактовата честота: Пропускане на някои събития.

-Ограничаване на времето за изпълнение: При итеративни алгоритми.

-Опашка с краен размер: Заявките, които не могат да се обработят веднага, се поставят в опашка; когато се освободи ресурс, се обработва следващата заявка; когато се напълни опашката, заявките се отказват.

-Управление на ресурсите

-Паралелна обработка: Оптимизация на времето чрез паралелна обработка на заявки.

-Излишък на данни/процеси: Cache, load-balancing.

-Включване на допълнителни ресурси: Повече и по-бързи процесори, памет, диск, мрежа.

-Арбитраж на ресурсите

Scheduling: Когато има недостиг на ресурси (т.е. спор за тях), рябва да има институция, която да решава (т.е. да извършва арбитраж) кое събитие да се обработи с предимство. Това се нарича scheduling. Някои от основните scheduling алгоритми са:

-FIFO: Всички заявки са равноправни и се обработват подред.

-Fixed Priority: на различните заявки се присвоява различен фиксиран приоритет; пристигащите заявки се обработват по реда на техния приоритет. Присвояването става съгласно: Семантичната важност; Изискванията за навременност; Изискванията за честота;

-Dynamic Priority: Последователно или според изискванията за навременност.

-**Статичен scheduling** – времената за прекъсване и реда за получаване на ресурси е предварително дефиниран.