எளிய தமிழில்
சாப்ட்வேர் டெஸ்டிங்

கி.முத்துராமலிங்கம்

எளிய தமிழில்
சாப்ட்வேர் டெஸ்டிங்

கி.முத்துராமலிங்கம்

நூல் : எளிய தமிழில் சாப்ட்வேர் டெஸ்டிங்
ஆசிரியர் : கி.முத்துராமலிங்கம்

அட்டைப்படம் : லெனின் குருசாமி
மின்னஞ்சல் : guruleninn@gmail.com

மின்னூலாக்கம் : அ.ஷேக் அலாவுதீன்
தமிழ் இ சர்வீஸ், tamileservice17@gmail.com

வெளியீடு : FreeTamilEbooks.com

பொருளடக்கம்

முன்னுரை

நான் மென்பொறியாளன் ஆன கதையே விந்தையான ஒன்று. படித்தது பொறியியலும் தொழில்நுட்பமும் என்றாலும் 'ஏட்டுச்சுரைக்காய் கறிக்கு உதவாது' என்ற பழமொழிக்குப் பொருத்தமானதாகத் தான் என்னுடைய கல்லூரி வாழ்க்கை இருந்தது. எதற்குப் படிக்கிறோம், என்ன படிக்கிறோம் என்றே தெரியாமல் படித்த படிப்பை எதற்குப் படித்தோம், என்ன படித்தோம் என்று பின் நாட்களில் வருந்திய நாட்களும் உண்டு. ஆனாலும் மென்பொருள் துறையிலேயே காலம் கழிந்து கொண்டிருந்தது.

அப்படிப்பட்ட காலத்தில் தான் சாப்ட்வேர் டெஸ்டிங் பற்றிய தொடரைக் கணியத்தில் எழுதத் தொடங்கினேன். எத்தனையோ பெரிய, பெரிய மென்பொறியாளர்கள் இருக்கும் தமிழ்த்திருநாட்டில் யாரும் எழுதவில்லை என்பதால் எழுதத் தொடங்கியதைத் தவிர, வேறு எந்த நோக்கமும் என்னிடம் இல்லை. நீண்ட நாட்களுக்கு முன்னரே, ஆர்வமாக எழுதத் தொடங்கிய இந்தத் தொடர், இடையிடையே வந்த சில சின்னச் சின்ன தடைகளால் முடையாகி விடுமோ என்று நினைத்திருந்தேன். அப்படி உடையாமல் விடையாக உங்கள் கைகளில் தவழ்கிறது இந்த மின் நூல்!

ஒவ்வொரு முறை அலைபேசியில் பேசும் போதும் 'அந்தத் தொடரை முடிச்சிடுங்களேன்' என்று அன்போடு கேட்பார் கணியம் சீனிவாசன். நானும் ஒவ்வொரு முறையும் 'முயல்கிறேன்', ‘தொடங்குகிறேன்', ‘முடிக்கிறேன்' என்று தட்டாமல் தட்டிக் கழித்துக் கொண்டிருந்தேன். கடந்த சில நாட்களாக என்னையும் அறியாமல் ஒரு வேகம் குடி கொண்டிருந்தது. எப்படியாவது கணியத்தில் தொடரை நல்லபடியாக முடித்து விடவேண்டும் என்று மனம் இடையறாது சொல்லிக் கொண்டிருந்தது. அந்த வேகத்தின் விளைவையே, நீங்கள் நூலாகப் பார்க்கிறீர்கள்.

தொடராக வந்த போது, எழுத்திலும் எதிரிலும் வந்து வாழ்த்திய வாசகர்களுக்கு நன்றிகள் பல! உங்களுடைய வாழ்த்துகள், பல நேரங்களில் உந்துதலாக இருந்திருக்கின்றன. தொடராக வந்த அப்பொழுதும் மின் நூலாக வரும் இப்பொழுதும் வாய்ப்பு வந்த கணியத்திற்கு எப்பொழுதும் என் நன்றிகள்!

எத்தனையோ வல்லவர்கள் இருக்க, என்னையும் ஒரு கருவியாகப் பயன்படுத்தி எழுத்துகளைக் கொண்டு வந்த - எல்லாம் வல்ல இறைவனை, தென்னாடுடைய சிவனை நெஞ்சார நினைக்கிறேன். நன்றியால் நனைக்கிறேன்.

31 அக்டோபர் 2019
மாலை 4.30

தோழமையுடன்
கி. முத்துராமலிங்கம்
muthu1809@gmail.com

சாப்ட்வேர் டெஸ்டிங் என்றால் என்ன?

சாப்ட்வேர் டெஸ்டிங் என்பதைப் பார்ப்பதற்கு முன்னர் 'டெஸ்டிங்' என்றால் என்ன என்று பார்த்து விடுவோம். 'டெஸ்டிங்' என்றால் என்ன? சோதிப்பது, ஆய்ந்து பார்ப்பது என்று சொல்லலாம். சோதிப்பது என்றால் எதைச் சோதிப்பது? பள்ளிக்கூடத்தில் 'டெஸ்ட்' (தேர்வு) என்று வைக்கிறார்கள். அங்கே என்ன சோதிக்கிறார்கள்? ஆசிரியர் கற்றுக்கொடுத்த பாடம் முழுவதும் மாணவர்களைப் போய்ச் சேர்ந்திருக்கிறதா என்று சோதிக்கிறார்கள். அந்தச் சோதனைக்கு மொத்தம் நூறு மதிப்பெண் வைத்துக்கொள்கிறார்கள். அதில் நூற்றுக்கு மாணவர்கள் எவ்வளவு மதிப்பெண் வாங்குகிறார்களோ அந்த அளவு சோதனை (டெஸ்ட்) வெற்றி என்று சொல்கிறார்கள். அப்படித் தானே!

அதாவது, மாணவர்கள் நூறு சதவீதம் படித்திருக்க வேண்டும் என்று ஆசிரியர் எதிர்பார்க்கிறார். அந்த எதிர்பார்ப்பை மாணவர்கள் எந்த அளவு நிறைவேற்றுகிறார்கள் என்பதைப் பொறுத்து 60, 70, 80 என மதிப்பெண்கள் கொடுக்கிறார். எனவே டெஸ்டிங்கின் வெற்றி, தோல்வி என்பது எந்த அளவு எதிர்பார்ப்பை நிறைவு செய்கிறோம் என்பதைப் பொறுத்து அமைகிறது. 100 க்கு 100 எடுத்தால் வெற்றி!

இதே போல் தான் சாப்ட்வேர் டெஸ்டிங் என்பதும்! நீங்கள் ஒரு கணினி நிறுவனம் நடத்திக்கொண்டிருக்கிறீர்கள் என்று வைத்துக் கொள்வோம். உங்களிடம் ஒரு ஒரு வாடிக்கையாளர், "கணியம்.காம் போலவே எனக்கு ஒரு இணையத்தளம் வேண்டும்" என்று கேட்கிறார் என்று வைத்துக்கொள்ளுங்கள். நீங்களும் உடனடியாக உங்கள் நிறுவன ஊழியர்களை வைத்து இணையத்தளத்தை வடிவமைக்கத் தொடங்கி விடுவீர்கள். இந்த இணையத்தளம் வாடிக்கையாளர் கேட்ட எல்லாச் செயல்பாடுகளையும் நிறைவு செய்கிறதா - என்று வாடிக்கையாளரிடம் மென்பொருளை ஒப்படைப்பதற்கு முன்பு நன்றாகச் சோதிப்பது தான் சாப்ட்வேர் டெஸ்டிங்! வாடிக்கையாளர் விரும்பிய எல்லாவற்றையும் நிறைவு செய்தால் டெஸ்டிங் வெற்றி! இல்லையெனில் தோல்வி! அவ்வளவு தான்!

மேல் உள்ள எடுத்துக்காட்டில் நாம் ஓர் இணையத்தளத்தை எடுத்துக் கொண்டோம். பொதுவாக, சாப்ட்வேர் என்பது இணையத்தளம் மட்டுமல்லாது பயர்பாக்ஸ் போல நிறுவி இயங்கும் மென்பொருளாகவோ, அலைபேசிகளில் இயங்கும் 'ஆப்' செயலியாகவோ எனப் பல வகைகளில் இருக்கலாம். இவற்றில் எதுவாக இருந்தாலும், அந்த மென்பொருள் எந்த நோக்கத்திற்காக உருவாக்கப்பட்டதோ அந்த நோக்கத்தை நிறைவு செய்கிறதா என்று பார்ப்பது தான் சாப்ட்வேர் டெஸ்டிங் ஆகும்.

நாம் உருவாக்கி ஒப்படைத்த மென்பொருளில், வாடிக்கையாளர் கேட்ட எல்லாச் செயல்பாடுகளும் எதிர்பார்த்தபடி இயங்கினால், வாடிக்கையாளர் 'நல்ல மென்பொருள்', 'தரமான மென்பொருள்' என்று மற்றவர்களிடம் பரிந்துரைப்பார். தரமான மென்பொருள் மூலம் வாடிக்கையாளரும் வளர்வார். நம்முடைய வியாபாரமும் வளரும்! இப்படி 'தரமான

மென்பொருளை' உருவாக்குவதற்குப் பயன்படுத்தும் உத்தி தான் 'சாப்ட்வேர் டெஸ்டிங்' ஆகும்.

சரி! 'தரம்' என்றால் என்ன? இரண்டாயிரம் ரூபாய்க்கு அலைபேசி வாங்குபவரும் தரமான அலைபேசி என்கிறார்! இருபதாயிரம் கொடுத்து வாங்குபவரும் தரமான அலைபேசி என்கிறார். அப்படியென்றால் 'தரம்' என்பதை எப்படி அளவிடுவது? தரமான மென்பொருள் என்று எப்படி வரையறுப்பது? தரம் என்பதன் அளவுகோல் விலையா? வெளித்தோற்றமா? வேறு எதுவுமா? அடுத்த பதிவில் பார்ப்போம்.

சாப்ட்வேர் டெஸ்டிங் - தரம் என்றால் என்ன?

தரம் என்றால் என்ன என்னும் கேள்வியுடன் முந்தைய பதிவை முடித்திருந்தோம். யோசித்துப் பார்த்தீர்களா? தரம் என்று எதைச் சொல்வது? விலை அதிகமாக உள்ள ஒரு பொருளைத் தரமானது என்று சொல்லலாமா? பொது நிலையில் அது சரி என்று தோன்றினாலும் உண்மை அதுவாக இருக்காது. விலை அதிகம் என்பதோடு தரமும் இல்லாத பொருட்கள் ஏராளம் சந்தையில் கிடைக்கின்றன. சரி! குறைந்த விலையில் கிடைக்கும் பொருட்கள் தரமற்றவை என்று சொல்ல முடியுமா? அப்படியும் வரையறுத்துச் சொல்ல முடியாது. ஏனென்றால், தரமான சில பொருட்கள் குறைந்த விலையில் நமக்குக் கிடைக்கின்றன. சரி! இதையும் விட்டு விடலாம்.

இலவசமாகக் கிடைக்கும் பொருட்களைத் தரமற்றவை என்று ஒதுக்கி விடலாமா? அப்படியும் சொல்லிவிட முடியாது. ஏனென்றால், விலை கொடுத்து வாங்கும் பல மென்பொருட்களை விட, இலவசமாகக் கிடைக்கும் வி எல் சி மீடியா பிளேயர், பயர்பாக்ஸ், லினக்ஸ் எனக் கட்டற்ற பல மென்பொருள்கள் தரமாக இருக்கின்றன. எனவே, இதனால் சகல மாணவர்க்கும் அறிவிப்பது என்னவென்றால், விலைக்கும் தரத்திற்கும் வரையறுக்கப்பட்ட எந்தத் தொடர்பும் கிடையாது.

சரி! ஒரு பொருளைத் தரமான பொருள் என்று எப்போது சொல்வீர்கள்? ஒரு பொருள் எல்லா எதிர்பார்ப்புகளையும் நிறைவு செய்தால் அதைத் தரமான பொருள் என்று சொல்லலாம். அப்பொருளில் சிற்சில குறைபாடுகள் இருக்கலாம்; தவறில்லை. ஆனால் வாடிக்கையாளர் எதை எதிர்பார்த்து அப்பொருளை வாங்குகிறாரோ அந்த முதன்மையான எதிர்பார்ப்பு நிறைவு செய்யப்பட வேண்டும். இந்த எதிர்பார்ப்புகள் ஒவ்வொரு வாடிக்கையாளருக்கும் மாறலாம்.

நீங்கள் ஓர் அலைபேசி வாங்குகிறீர்கள் என்று வைத்துக் கொள்வோம். நீங்கள், உங்கள் அம்மாவிற்கோ அப்பாவிற்கோ வாங்கும் போது உங்களுடைய முதன்மை எதிர்பார்ப்பு - பயன்படுத்த எளிதாக இருக்க வேண்டும் என்பதாக இருக்கலாம். அதே அலைபேசியைக் கல்லூரியில் படிக்கும் உங்கள் தம்பிக்கோ தங்கைக்கோ வாங்கினால், எதிர்பார்ப்பு மாறிவிடும். பயன்படுத்தக் கடினமாக இருந்தாலும் பரவாயில்லை - இணையம் முதலிய எல்லா வசதிகளும் இருக்க வேண்டும் என்பதாக இருக்கும். சரிதானே! எல்லா வசதிகளுடன் கூடிய அலைபேசி வாங்கும் போது அதில் பேட்டரி நீண்ட நேரம் நிற்காமல் போகலாம். ஆனால் பேட்டரி நிலைத்து நிற்க வேண்டும் என்பது உங்களுடைய அடிப்படைத் தேவையில்லை. எனவே ஏற்றுக்கொள்ளத் தயங்க மாட்டீர்கள். சரி தானே!

எனவே, தரம் என்பதற்கு

- வாடிக்கையாளரின் தேவைகளை நிறைவு செய்வது
- சரியான பொருளுக்குச் சரியான விலை கொண்டு சரியான வசதிகளைக் கொடுப்பது
- குறைகள் ஏதுமின்றிப் பொருள் இருப்பது (அல்லது கண்ணுக்குத் தெரியாத குறைகளை மட்டும் கொண்டிருப்பது)

எனப் பல விளக்கங்களைக் கொடுக்கலாம்.

இந்தத் தரம், வெளியாகும் மென்பொருளில் இருக்கிறதா என்று சோதிப்பது தான் சாப்ட்வேர் டெஸ்டிங்கின் அடிப்படையாகும்.

சரி! டெஸ்டிங் ஏன் செய்கிறோம் என்னும் அடிப்படையைப் புரிந்து கொண்டீர்கள்! சாப்ட்வேர் என்றால் என்ன? அதில் டெஸ்டிங்கின் அடிப்படைகள் என்னென்ன? பேசுவோம்!

சாப்ட்வேர் டெஸ்டிங் - சாப்ட்வேர் என்றால் என்ன?

முந்தைய பதிவுகளில் டெஸ்டிங் என்றால் என்ன என்பது பற்றியும் சாப்ட்வேர் டெஸ்டிங்கின் அடிப்படை நோக்கம் பற்றியும் பார்த்திருந்தோம். இப்போது நம் முன்னால் நிற்கும் கேள்வி - சாப்ட்வேர் (மென்பொருள்) என்றால் என்ன? இதெல்லாம் என்ன கேள்வி? இதோ சொல்கிறேன் - கணினிக்கு ஏதோ ஓர் உள்ளீட்டைக் கொடுத்து தேவைப்படும் பதிலை எடுத்துக்கொள்வது - சரிதானா? என்கிறீர்களா! நூற்றுக்கு நூறு சரி! மென்பொருள் என்பது அது தான்! அதைத் தான் நாம் சோதிக்கப் (டெஸ்டிங்) போகிறோம்.

மென்பொருளை, லினக்சில் இயங்கும் மென்பொருள், விண்டோஸ் மென்பொருள், மேக் மென்பொருள் என்று இயங்கு தள அடிப்படையிலோ, ஜிமெயில், பேஸ்புக், டக்டக்கோ என்று பயன்பாட்டு அடிப்படையிலோ பல வகைகளில் பிரிக்கலாம். எந்த வகைகளில் பிரித்தாலும் நம்முடைய வேலை அவற்றைச் சோதிப்பது! சோதித்து வாடிக்கையாளரைப் பாதிக்கும் தவறுகளை உருவாக்குநரிடம் (டெவலப்பரிடம்) எடுத்துச் சொல்வது! இவை தாம் சாப்ட்வேர் டெஸ்டராக நம்முடைய வேலை! எனவே நம்முடைய வேலையின் அடிப்படையில் சாப்ட்வேரை இரண்டு பெரும் பிரிவுகளில் பிரித்து விடலாம்.

1. கணினியில் நிறுவி (இன்ஸ்டால் செய்யப்பட்டு) இயங்கும் மென்பொருள்
2. இணைய வசதியுடன் இயங்கும் மென்பொருள்

அடிப்படையாக இந்த இரண்டு வகை மென்பொருள்களையும் சோதிக்கத் தெரிந்தாலே நாம் 'டெஸ்டர்' என்று சொல்லிக் கொள்ளலாம். இவற்றை எப்படிச் சோதிப்பது? பார்ப்போம்.

1. கணினியில் நிறுவி (இன்ஸ்டால் செய்யப்பட்டு) இயங்கும் மென்பொருள்

இந்தத் தலைப்பை வைத்தே இவ்வகை சாப்ட்வேரை எப்படிச் சோதிக்கலாம் என்று தெரிந்து கொள்ளலாம். ஆமாம்! நாம் செய்ய வேண்டிய முதல் சோதனை - மென்பொருளை நம்மால் எளிதாக கணினியில் நிறுவ முடிகிறதா என்பது தான்!

இயங்குதளப் பொருத்தச் சோதனை ('OS Compatibility Test')

- உருவாக்கப்பட்டுள்ள மென்பொருள் லினக்ஸ் தளத்தில் இயங்கும் படி உருவாக்கப்பட்டுள்ளதா, விண்டோசுக்காக உருவாக்கப்பட்டுள்ளதா, ஆண்டிராய்டுக்கா எனப் பார்த்து அந்தந்த இயங்குதளத்தில் நிறுவிச் சிக்கல் வருகிறதா எனப் பார்க்க வேண்டும்.
- சில மென்பொருள்கள், இயங்குதளத்தின் குறிப்பிட்ட பதிப்புகளுக்கு மட்டும் இயங்குமாறு உருவாக்கப்பட்டிருக்கும். எடுத்துக்காட்டாக, விண்டோஸ் 7 அல்லது 8 இல் மட்டும் இயங்குமாறு சில மென்பொருட்கள் உருவாக்கப்பட்டிருக்கும். அந்நிலையில் அவற்றைப் பொருத்தமான (விண்டோஸ் 7, 8) இயங்குதளங்களில் மட்டும் நிறுவி, நிறுவலில் ஏதாவது சிக்கல் வருகிறதா எனச் சோதிக்க வேண்டும்.

இச்சோதனைக்கு இயங்குதளப் பொருத்தச் சோதனை ('OS Compatibility Test') என்று பெயர்.

நிறுவல் - உயர்த்தல் சோதனைகள்:

பொதுவாக லிபர் ஆபிஸ் போல ஒரு மென்பொருளை விண்டோசில் நிறுவுகிறீர்கள் என்றால், இயல்பாகவே அது c:\Program Files\ என்பது போல ஓர் இயல்பு நிலை அமைவிடத்தில் நிறுவப்பட்டு விடும். இல்லை எனக்கு c:\Program Files\ வேண்டாம்; நான் வேறு இடத்தில் நிறுவுகிறேன் என்றால் நீங்கள் இடத்தை மாற்றிக் கொள்ளலாம். இந்த இருவகை நிறுவலையும் நாம் சோதிக்க வேண்டும்.

- இயல்பு நிலை இடத்தில் (டீஃபால்ட் லொகேசன்) மென்பொருளை நிறுவ முடிகிறதா?
- வேறு இடத்தில் மென்பொருளை நிறுவ முடிகிறதா?
- இப்படி இரு நிலைகளும் முடிகிறது என்றால் நிறுவிய பின்னர், வேறு பதிப்புகளுக்குத் தரம் உயர்த்த முடிகிறதா எனச் சோதிக்க வேண்டும்.

எடுத்துக்காட்டாக, லிபர் ஆபிஸ் 4.4.2 ஐ நிறுவியிருக்கிறீர்கள் என்றால், நிறுவிய பின்னர் அதை 4.4.3.2 க்கு உயர்த்த முடிகிறதா எனச் சோதிக்க வேண்டும். இச்சோதனைக்குத் 'தரம் உயர்த்து சோதனை' ('Upgrade Testing') என்று பெயர்.


LibreOffice 4.4.3.2 The Document Foundation 02-Jun-15 1.16 GB 4.4.3.2

இச்சோதனையைச் செய்த பின்னர், நாம் நிறுவியிருக்கும் மென்பொருளின் பதிப்பு சரியானதா எனப் பார்த்துக்கொள்ள வேண்டும்.

நீக்கல் சோதனை:

நிறுவல் சோதனை போல, நீக்கல் சோதனையும் முக்கியமானது தான்! அதென்ன நீக்கல் சோதனை என்கிறீர்களா? உங்கள் ஊகம் சரிதான்! நிறுவப்பட்ட மென்பொருளைக் கணினியில் இருந்து நீக்க (அல்லது தூக்க) முடிகிறதா, என்று பார்ப்பது தான்! பயனாளர் 'இந்த மென்பொருள் தனக்குத் தேவையில்லை' என்று நினைத்தால் சிக்கல் ஏதும் இல்லாமல் நீக்க ஏதுவாக மென்பொருள் வடிவமைக்கப்பட்டிருக்க வேண்டும். அப்படி நீக்கிச் சோதிப்பது நீக்கல் சோதனை ('Uninstallation Testing') ஆகும்.

என்ன நீங்கள் - நிறுவப்பட்ட மென்பொருளை எப்படிச் சோதிப்பது என்று சொல்வீர்கள் என்று பார்த்தால் - நிறுவல், உயர்த்தல், நீக்கல் என்று ஆக்கல், ஒடுக்கல், அழித்தல் என்று கடவுளுக்கு மூன்று தொழில் சொல்வது போலச் சொல்கிறீர்களே! என்று நினைக்கிறீர்களா? நீங்கள் நினைப்பது சரி தான்! சாப்ட்வேர் டெஸ்டிங் என்பதில் நிறுவப்பட்டு இயக்கப்படும் மென்பொருள்களைச் சோதிக்கும் உத்திகளைப் பிறகு பார்ப்போம். இப்போதைக்கு, எந்த மென்பொருளாக இருந்தாலும் எப்படி அடிப்படையாகச் சோதிக்க வேண்டும் என்னும் பொதுநிலையைத் தெரிந்து கொண்டோம். அவ்வளவு தான்!

சோதனை ஓட்டத்தை('Trial Version')ச் சோதிப்பது:

சில மென்பொருட்களைச் சோதனைக் காலமாக 30 நாட்கள் மட்டுமோ இலவசமாகப் பயன்படுத்திப் பார்ப்பதற்கு நிறுவனங்கள் வழங்கும். (சில நிறுவனங்கள் 15 நாட்கள் கொடுக்கின்றன.) அப்படிப்பட்ட மென்பொருட்களை

- முப்பது நாட்கள் இயங்குகின்றனவா?

முப்பது நாட்கள் சோதனையைக், கணினியின் தேதியை முப்பது நாட்கள் மாற்றிச் செய்ய முடியும்.

- முப்பத்தோராவது நாளில் இருந்து 'மென்பொருள் இயங்காது' என்னும் செய்தி காட்டப்பட்டுகிறதா?
- கணினியின் தேதியை முன் தேதியிட்டு அமைப்பதன் மூலம் சோதனை ஓட்டத்திற்குக் கொடுக்கப்பட்ட நாட்களில் ஏதாவது மாற்றம் ஏற்படுகிறதா?

என்பனவற்றைப் பார்க்க வேண்டும்.

இந்த அடிப்படையுடன், நிறுவப்பட்ட மென்பொருளின் இயக்கத்தையும் முழுமையாகச் சோதிக்க வேண்டும். இந்த இயக்கச் சோதனை மிகவும் முக்கியமானது!

இதெல்லாம் சரி! இணைய வசதியுடன் இயங்கும் மென்பொருட்களை எப்படிச் சோதிப்பதன் அடிப்படைகள் என்னென்ன? என்று தெரிந்து கொள்ள வேண்டுமே! என்று நினைக்கிறீர்களா! அடுத்த பதிவு அதைப் பற்றித் தான்!

சாப்ட்வேர் டெஸ்டிங் - இணைய வழி இயங்கும் மென்பொருள் சோதனைகள்

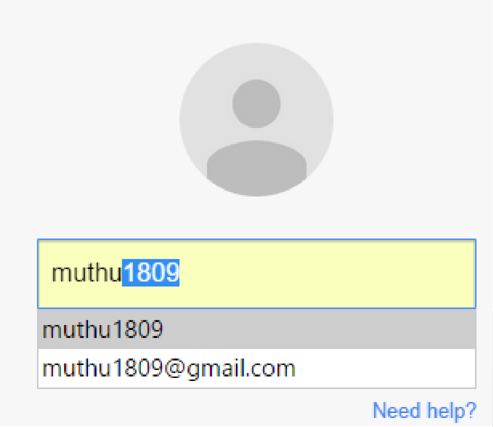
என்ன இது? தலைப்பே புரியவில்லை என்று தோன்றுகிறதா? கவலைப்படாதீர்கள்! புரியும்படிப் பார்த்து விடலாம்! இணையம் இல்லாமல் எந்தெந்த மென்பொருள்கள் (சாப்ட்வேர்) எல்லாம் இயங்காதோ, அவையெல்லாம் இணைய வழி இயங்கும் மென்பொருள்கள் தாம்! அப்படியானால், கணியம்.காம் என்பது இணையவழி இயங்கும் மென்பொருள் - சரிதானா என்கிறீர்களா? நூற்றுக்கு நூறு சரிதான்!

ஓர் இணையத்தளத்தையோ, வலைப்பூவையோ சோதிக்க வேண்டுமானால் அடிப்படையில் எதையெல்லாம் கவனம் எடுத்துப் பார்க்க வேண்டும் - அவற்றைத் தான் இப்போது பார்க்கப் போகிறோம்.

1. பயன்பாட்டுச் சோதனை ('Usability Test')

இணையத்தளத்தைப் பொருத்தவரை, பயன்படுத்துவதற்கு எளிதாக இருக்க வேண்டும். எளிமையான பயன்பாட்டு முறைக்காகப் பல்வேறு வழிமுறைகளை இணையத்தளங்கள் பின்பற்றி வருகின்றன. அந்த வழிமுறைகள் நாம் சோதிக்கும் இணையத்தளத்திலும் இருக்கின்றனவா என்று பார்த்துக் கொள்ள வேண்டும். சில எளிய எடுத்துக்காட்டுகளை உங்களுக்குக் காட்டுகிறேன்.

அ. ஜிமெயில் போன்ற இணையத்தளங்களில் முகப்புப் பக்கத்தில், நீங்கள் ஏற்கெனவே கொடுத்திருந்த பயனர் பெயர் கீழ்வரிசையில் தெரியும். நீங்கள் அந்த வரிசையில் இருந்து பயனர் பெயரைத் தெரிந்தெடுத்துக் கொள்ளலாம்.

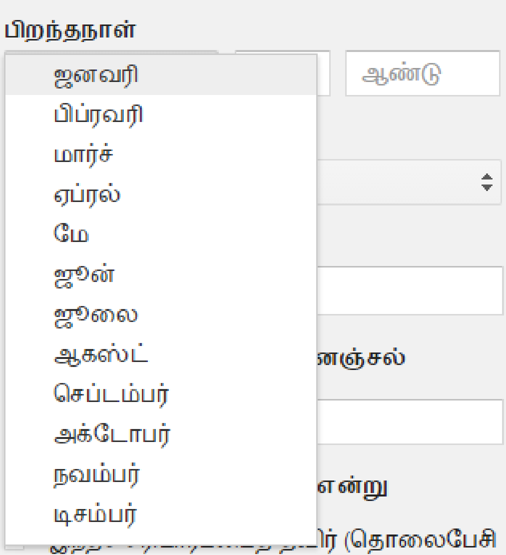


muthu1809

muthu1809

muthu1809@gmail.com

ஆ. கடவுச்சொல்லை நினைவில் கொள்க ('Remember Password') என்னும் தேர்வைக் கொடுப்பது.



இ. விண்ணப்பப் படிவங்களில் நாடுகளின் பெயர்கள், மாதங்கள் மாநிலங்களின் பெயர்கள், ஆகியவற்றைக் கீழ் வரிசைப் பட்டியலில் தருவது.

2. இணைப்புச் சோதனை ('Link, Navigation Test')

தலைப்பை வைத்தே ஓரளவு ஊகித்திருப்பீர்கள்! உங்கள் ஊகம் சரி தான்! இணையத்தளச் சோதனையில் மிகவும் முக்கியமானது இது! அதே வேளையில் ரொம்ப எளிதானதும் கூட! இணையத்தளத்தை விட்டு விடுங்கள். ஓர் எடுத்துக்காட்டைப் பார்ப்போம். நீங்களும் உங்கள் நண்பரும் வண்டியில் நீண்ட தூரம் போய்ப் பார்க்க விரும்புகிறீர்கள். வண்டியை எடுத்துக் கொண்டு பல மைல் தூரம் போய் விட்டீர்கள். போன பிறகு அங்கிருந்து எப்படித் திரும்பி வருவது என்று தெரியவில்லை! ஏதாவது வழிகாட்டிப்பலகை, சாலையை ஒட்டி இருக்காதா, என்று பார்த்துக் கொண்டே வருவீர்கள் - அப்படித் தானே! அந்த வழிகாட்டிப் பலகை தான் உங்களைத் தொடங்கிய இடத்திற்கே கொண்டுவந்து சேர்க்க உதவும்! இதே தான் இணையத் தளத்திலும்!

1) நீங்கள் இணையத்தளத்தின் ஒரு பக்கத்தில் இருந்து எந்தெந்த இணைப்புகளையெல்லாமோ தொட்டுத் தொட்டு ஏதோ ஒரு பக்கத்திற்கு வந்து விட்டீர்கள்! வந்த பிறகு மீண்டும் முதல் பக்கத்திற்குப் போக வேண்டும் என்றால் என்ன செய்வது? சாலையில் வழிகாட்டிப் பலகை இருந்தது போல, இங்கும் வழிகாட்டிகள் இருக்க வேண்டாமா?

அப்படி வழிகாட்டிப் பலகையே இல்லை என்றால் நீங்கள் நிற்கும் இடத்தில் இருந்து வெளியேற முடியாது தவிப்பீர்கள். இணையத்தில் இப்படி இணைப்புகளே இல்லாமல் தனித்து நிற்கும் இணையப்பக்கங்களுக்குத் தனிமரங்கள் ('Orphan pages') என்று பெயர்! இப்படித் தனிமரங்கள் இல்லாமல் உங்கள் இணையத் தளத்தைப் பார்த்துக் கொள்ள வேண்டும்.

இணைப்புகள் இருந்தால், அந்த இணைப்புகள் சரியாக வேலை செய்ய வேண்டும். இணையத் தளத்தில் எந்த இணைப்பை (link) நீங்கள் தொடுகிறீர்களோ, அந்த இணைப்புக்கு நாம் போக வேண்டும். அப்படிப் போகவில்லை என்றாலோ, தவறான இணைப்புக்குப் போனாலோ தவறு! அதை உடனடியாகத் தெரியப்படுத்த வேண்டும்.

3. பாதுகாப்புச் சோதனை ('Security Testing')

இணையம் என்று வந்து விட்டாலே பாதுகாப்பு என்பது மிகவும் முக்கியமானது! தகவல்களைப் பாதுகாக்க முடியவில்லை என்றால் இணையத்தைப் பயன்படுத்தவே முடியாது! எனவே தான் பாதுகாப்புச் சோதனை முக்கியமானதாகக் கருதப்படுகிறது.

அ) சரியான பயனர் பெயர், கடவுச் சொல்லுக்குத் தான் தகவல்கள் பகிரப்படுகின்றனவா

ஆ) பயனர் நிலைக்கு ஏற்ப, தேவையான அனுமதிகள் கொடுக்கப்படுகின்றனவா? எ.கா. ஒரு பயனர், நிர்வாகியாக இருந்தால், இணையத்தளத்தில் தேவைப்படும் மாற்றங்களைச் செய்ய அவருக்கு அனுமதி கொடுக்கப்பட வேண்டும். அவரே சாதாரண பயனராக இருந்தால், பார்வை அனுமதி மட்டுமே போதுமானது.

இ) பயனர் பெயர், கடவுச் சொல் ஆகியவற்றைக் கேட்கும் இணையத்தளங்களின் எந்தப் பக்கத்தைத் திறந்தாலும் முதலில் பயனர் பெயர், கடவுச் சொல் ஆகியவற்றைக் கேட்கிறதா? என்று பார்ப்பது.

4. உலவிப் பொருத்தச் சோதனை ('Browser Compatibility Testing')

பயனர் எந்தெந்த உலவிகளை ('Browser')ப் பயன்படுத்துகிறாரோ, அந்தந்த உலவிகளில் இணையத்தளம் ஒழுங்காகத் திறக்கிறதா? எல்லாப் படங்களும் சரியாகக் காட்டப்படுகின்றனவா? என்பனவற்றைச் சோதிக்க வேண்டும். அடிப்படை நிலையில் பெரும்பாலானோர் பயன்படுத்தும் உலவிகளான பயர்பாக்ஸ், குரோம், இன்டர்நெட் எக்ஸ்புளோரர் ஆகியவற்றின் அண்மைப் பதிப்புகளில் (latest versions) இணையத்தளம் ஒழுங்காகச் செயல்படுகிறதா என்று பார்க்க வேண்டும்.
இப்போது அலைபேசிகள், சிறு கணினிகள் ஆகியன வந்து விட்டன. அந்தத் திரைகளிலும் இணையத் தளம் சரியாகத் திறக்கின்றதா என்று பார்க்க வேண்டும்.

5. தரவு சரிபார்ப்புச் சோதனை ('Validation Testing')

என்ன தலைப்பு இது என்று தலையைச் சொறிகிறீர்களா? எளிமையானது தான்! தரவு என்றால் பயனர் இணையத்தளத்தில் தரும் தகவல்! அந்தத் தகவல் சரியானதாக இருக்கிறதா, சரியான வடிவத்தில் இருக்கிறதா என்று பார்ப்பது தான் தரவு சரிபார்ப்புச் சோதனை!

- நீங்கள் கொடுக்கும் அலைபேசி எண், பத்து இலக்கங்களுடன் இருக்கிறதா என்று பார்ப்பது! அப்படி இல்லாவிட்டால் 'பத்து இலக்கங்கள் இல்லை- தயவு செய்து சரிபாருங்கள்' என்று செய்தி கொடுப்பது!
- ஒருவர் மின்னஞ்சலைக் கொடுத்தால் அது '@', '.com', '.co.in', '.org' என்பன போன்ற குறியீடுகளைக் கொண்டிருக்கிறதா என்று பார்ப்பது
- கடவுச்சொல் ('Password') கட்டம் காலியாக இருக்கக் கூடாது என்பது போன்ற சரி பார்ப்புகள்!
- ஆகியன தரவு சரிபார்ப்புச் சோதனைக்கான எளிய எடுத்துக்காட்டுகள்! சொன்னது போல் இச்சோதனை எளிதாகத் தானே இருக்கிறது?

6. செயல் திறன் சோதனை ('Performance Testing')

பல இணையத்தளங்கள் மிகச் சிறப்பாக வேலை செய்யும். ஆனால் இணையத்தளத்தைத் திறப்பதற்கே பல நிமிடங்கள் ஆகும். சில சமயங்களில் ஒரு குறிப்பிட்ட இணைப்பைத் திறப்பதற்குள் நமக்குப் போதும் போதும் என்றாகி விடும்! இதையெல்லாம் எதிர்கொண்டிருக்கிறீர்களா? இதைத் தான் செயல் திறன் சோதனை என்று சொல்கிறோம்.

நம்முடைய கணினி நிறுவனம் ஓர் இணையத்தளத்தை உருவாக்குவதாக வைத்துக்கொள்ளுங்கள். ஒரே நேரத்தில் பத்தாயிரம் பேர் அந்தத் தளத்தைப் பார்க்கும் போது தளத்தின் வேகம் எப்படி இருக்கிறது? இணையப்பக்கங்களைத் திறக்க எவ்வளவு நேரம் ஆகிறது? எல்லாப் படங்களும் ஒழுங்காகக் காட்டப்படுகின்றனவா? என்பன போன்றவற்றை ஆராய வேண்டும். இதுவே செயல்திறன் சோதனை ('Performance Test') என்று சொல்லப்படுகிறது.

ஒரே நேரத்தில் பத்தாயிரம் பேரா? எப்படிப் பத்தாயிரம் பேரைத் திரட்டி இணையத்தளத்தைச் சோதிப்பது? என்று நீங்கள் சிந்திக்கலாம்! அப்படிப் பத்தாயிரம் பேர் தளத்தைப் பார்ப்பது போன்ற சூழலை உருவாக்க ஏராளமான செயல்திறன் சோதனை மென்பொருட்கள் உள்ளன. எல்லா இடங்களிலும் நீங்கா இடம் பெற்றிருக்கும் கட்டற்ற மென்பொருட்கள் இங்கும் கிடைக்கின்றன. ஜேமீட்டர் ('JMeter') என்பது அவற்றுள் பலரும் அறிந்த கட்டற்ற மென்பொருள் ஆகும்.

நீங்கள் சொல்வதெல்லாம் புரிகிறது. ஆனால் ஒரு கணிப்பொறி நிறுவனத்தில் மென்பொருள் சோதனை (சாப்ட்வேர் டெஸ்டிங்) என்பது எப்போது தொடங்கும்? நான் சாப்ட்வேர் டெஸ்டிங் செய்ய என்னென்ன செய்ய வேண்டும்? மென்பொருள் சோதனையாளராக (டெஸ்டராக) மாற, புரோகிராமிங் அறிவு தேவையா? இப்படிப் பல கேள்விகள் இருக்கின்றனவா? அவற்றையெல்லாம் பார்ப்போம் - இனி வரும் பதிவுகளில்!

சாப்ட்வேர் டெஸ்டிங் - எங்கு தொடங்குவது?

சாப்ட்வேர் டெஸ்டிங்கின் அடிப்படைகளைப் பார்த்து விட்டோம். இப்போது நம் முன்னால் இருக்கும் கேள்வி - சாப்ட்வேர் டெஸ்டிங்கை எங்கு, எப்படித் தொடங்குவது? என்பது தான்!

ஒரு மென்பொருளைச் சோதிக்க வேண்டும் என்று நாம் விரும்பினால், முதலில் அந்த மென்பொருள் தயாராக இருக்க வேண்டும் அப்படித் தானே! எனவே,

- எப்படி ஒரு மென்பொருள் உருவாக்கப்படுகிறது?
- வாடிக்கையாளரிடம் இருந்து மென்பொருளுக்கான தகவல்களை எப்படிப் பெறுவது? யார் அந்தத் தகவல்களை வாங்கித் தருவார்கள்?
- வாங்கிய தகவல்களை வைத்துக் கொண்டு நம்முடைய நிறுவனம் (அதாவது மென்பொருளை உருவாக்கும் நிறுவனம்) என்ன செய்யும்?
- எப்போது நமக்கு டெஸ்டிங்கிற்குத் தருவார்கள்?

இப்படிப் பல கேள்விகள் இருக்கின்றன. சரி தானே!

இவற்றையெல்லாம் பற்றிப் பேசுவதற்கு முன்னர், ஒரு சிறிய கதையைப் பார்ப்போம்.

தமிழினிக்குத் திருமணம்:

நல்லூர் என்பது ஓர் அழகிய சிற்றூர். அந்தக் கிராமத்தில் அறிவொளி என்பவர் வாழ்ந்து வந்தார். அவருடைய ஒரே மகள் தமிழினி. தமிழினி சிறு வயதில் இருந்தே பேச்சிலும் எழுத்திலும் படு சுட்டி! அவளும் வளர்ந்து படித்து, திருமண வயதை எட்டினாள்.

அவளுக்குத் தகுந்த மணமகன் தேடும் வேலையைத் தொடங்கினார் அறிவொளி. கதிரவன் என்பவர் தகுந்த மணமகனாகப் படவே மகளிடமும் பேசி ஒப்புதல் வாங்கித் திருமண வேலைகளைத் தொடங்கினார். ஒரே மகளின் திருமணம் என்பதால் தடபுடலாக நடத்த விரும்பிய அறிவொளி, பெரிய திருமண மண்டபம் ஒன்றை வாடகைக்குப் பிடித்தார். 'கல்யாண வீடு என்றாலே சாப்பாடு ரொம்ப முக்கியம், வாய் வாழ்த்தாவிட்டாலும் வயிறு வாழ்த்தும்' என்று நினைத்த அறிவொளி, பல பேரிடம் போய் "நல்ல சமையல்காரர்கள் இருந்தால் சொல்லுங்கள், என் மகள் திருமணத்திற்குச் சமையலுக்குத் தேடிக் கொண்டிருக்கிறேன்" என்று சொல்லி வைத்தார்; பலரும் பல சமையல்காரர்கள் முகவரிகளைக் கொடுத்தார்கள்.

அவர்கள் ஒவ்வொருவரையும் அலைபேசியில் கூப்பிட்டு 'நீங்கள் நேரில் என்னுடைய வீட்டுக்கு வர முடியுமா? அடுத்த மாதம் என்னுடைய மகள் திருமணத்திற்குச் சமையலுக்கு ஆள் தேடிக்கொண்டிருக்கிறேன்' என்று சொன்னார். அவர் கூப்பிட்ட படி, ஒவ்வொரு சமையல்காரரும் வந்து,

1. எந்தத் தேதியில் சமைக்க வேண்டும்?
2. எத்தனை பேருக்குச் சமைக்க வேண்டும்?

3. என்னென்ன சமைக்க வேண்டும்? சைவமா? அசைவமா?
4. எத்தனை வேளைகள் சமைக்க வேண்டும்?
5. இதற்கெல்லாம் அறிவொளி எவ்வளவு செலவு பண்ணலாம் என்று திட்டமிட்டிருக்கிறார்?
6. அவர் கேட்கும் உணவு வகைகளுக்கு எவ்வளவு ஆகும்? தங்களால் அவற்றைச் செய்ய முடியுமா?

என்பன போன்ற விவரங்களைப் பேசுவார்கள் அல்லவா? பேசிய விசயங்களை எல்லாம் எழுதி ஆவணப்படுத்தித் திட்டமிட்டு வேலை செய்யத் தொடங்குவார்கள் அல்லவா? எனவே இதை ஒரு திட்டமிட்ட பணி (வேலை) என்று சொல்லலாம். இதைத் தான் - திட்டப்பணி (புராஜெக்ட்) என்று சொல்கிறார்கள்.

வாடிக்கையாளர் தேவை ஆவணம்

அறிவொளியின் மகள் திருமணத்திற்குச் சமையல் வேலை என்பதற்குப் பதிலாக, பிரபல நகைக்கடை ஒன்றிற்கு இணையத்தளம் என்று வைத்துக்கொள்ளுங்கள். இந்த இணையத்தளத்தை வடிவமைக்கும் வேலை எப்படித் தொடங்கும்?

1. எந்தத் தேதியில் இணையத்தளம் வேண்டும்?
2. ஒரு நேரத்தில் அதிக பட்சம் எத்தனை பேர் பார்க்குமாறு தளம் அமைய வேண்டும்?
3. என்னென்ன உருப்படிகளை(வீடியோ, புகைப்படங்கள், கட்டுரை, செய்தி)த் தளத்தில் காட்ட வேண்டும்?
4. எத்தனை ஆண்டுகளுக்கு நிர்வகிக்க வேண்டும்?
5. இதற்கெல்லாம் அந்த நகைக்கடைக்காரர்கள் எவ்வளவு செலவு பண்ணலாம் என்று திட்டமிட்டிருக்கிறார்கள்?
6. என்ன தொழில்நுட்பம் பயன்படுத்த விரும்புகிறார்கள்?

என்பன போன்ற விவரங்களை எல்லாம் மென்பொருள் நிறுவனம் - நகைக்காரரிடம் கேட்டுத் தெரிந்து கொள்ள வேண்டும்; தெரிந்து அவற்றையெல்லாம் ஆவணப் படுத்தி வைத்துக் கொள்ள வேண்டும்.

ஏன் ஆவணப்படுத்த வேண்டும்?

ஏன் ஆவணப்படுத்துகிறோம் என்பதை எளிதாகச் சொல்லி விடலாம் - ஆவணப்படுத்தாமல் விட்டு விட்டால், பிறகு நகைக்கடைக்காரர் இப்படிக் கேட்டாரா, அப்படிக் கேட்டாரா, இதைச் செய்யச் சொன்னாரா? சொல்லவில்லையா? எனப் பல குழப்பங்கள் வரும். அவற்றையெல்லாம் தவிர்க்கவே ஆவணப்படுத்துகிறோம். இந்த ஆவணத்தில் வாடிக்கையாளரின் தேவைகள் எல்லாம் சொல்லப்பட்டிருக்கும். எனவே இதை, வாடிக்கையாளர் தேவை ஆவணம் ("பிசினஸ் ரெக்கைர்மென்ட் ஸ்பெசிபிகேசன் - Business

Requirement Specification (BRS)”) என்று சொல்வார்கள். இந்த ஆவணத்தின் அடிப்படையில் தான் மென்பொருளை உருவாக்கத் தொடங்குவார்கள்.

இது வரை நாம் பார்த்ததில் இருந்து பல சந்தேகங்களும் கேள்விகளும் வந்திருக்கும்.

- நகைக்கடைக்காரரைப் போய் நம்முடைய மென்பொருள் நிறுவனத்தில் இருந்து யார் பார்ப்பார்கள்?
- இந்த ஆவணப்படுத்தும் வேலையை வாடிக்கையாளர் செய்வாரா, வாடிக்கையாளரைப் போய்ப் பார்க்கும் நம்முடைய ஆள் செய்வாரா?
- அந்த ஆவணத்தில் என்னென்ன விவரங்கள் இருக்க வேண்டும்?
- உருவாக்கப்படும் ஆவணம் - சரியாகத் தான் உருவாக்கப்பட்டிருக்கிறது என்பதற்கான உறுதி என்ன?

என்பன போன்ற பல கேள்விகள்! தொடர்ந்து பார்ப்போம்!

சாப்ட்வேர் டெஸ்டிங் - சாப்ட்வேர் எங்கு தொடங்குகிறது?

முந்தைய பதிவில் நாம் பார்த்த நகைக்கடைக்கு இணையத்தளம் என்னும் எடுத்துக்காட்டின் அடிப்படையில் இதைக் கொஞ்சம் பார்ப்போம்! முந்தைய பதிவைப் படிக்காதவர்கள் தயவுசெய்து அதைப் படித்து விட்டு இப்பதிவிற்கு வாருங்கள்!
இதைப் பார்ப்பதற்கு முன்னர் அறிவொளியின் மகள் தமிழினியின் திருமணம் பற்றிப் பேசிக்கொண்டிருந்தோமே! அந்தப் பேச்சை முடித்து விடுவோமே! தமிழினியின் திருமணத்திற்குச் சமையல் வேலை செய்ய ஆள் தேடிக் கொண்டிருந்தோமே! நினைவிருக்கிறதா?
ஒவ்வொரு சமையல்காரரும் வந்து, எந்தத் தேதியில், எத்தனை பேருக்கு, என்னென்ன, எத்தனை வேளைகள் , எவ்வளவு செலவில் சமைக்க வேண்டும்? என்பன போன்ற விவரங்களைப் பேசுவார்கள் என்று பார்த்தோமே! சரி! நினைவுக்கு வந்திருக்கும்.
இப்போது உங்களிடம் சில கேள்விகள்!

- மேலே உள்ள கேள்விகளுக்கெல்லாம் சரியான பதிலைச் சமையல்காரர்களிடம் யாரால் சொல்ல முடியும்?
- சமையல்காரர்கள் கேட்கும் கேள்விகளுக்கு யாரால் சரியான பதில் சொல்ல முடியும்?

சொல்லுங்களேன்.
இதென்ன சாதாரண கேள்விகள் தானே! அறிவொளி தான் முடிவெடுப்பார்; அவர்தாம் சமையல்காரர்கள் கேள்விகளுக்குப் பதில் சொல்வார் என்கிறீர்களா? சரியாகச் சொன்னீர்கள்.
இதே போல் தான் நகைக்கடைக்காரர் - மென்பொருள் நிறுவன எடுத்துக்காட்டும்!
நகைக்கடைக்காரர் - மென்பொருள் நிறுவன எடுத்துக்காட்டில் - முந்தைய பதிவின் கடைசியில், நம் முன்னால் நிற்கும் கேள்விகள்:

- நகைக்கடைக்காரரைப் போய் நம்முடைய மென்பொருள் நிறுவனத்தில் இருந்து யார் பார்ப்பார்கள்?
- இந்த ஆவணப்படுத்தும் வேலையை வாடிக்கையாளர் செய்வாரா, வாடிக்கையாளரைப் போய்ப் பார்க்கும் நம்முடைய ஆள் செய்வாரா?
- அந்த ஆவணத்தில் என்னென்ன விவரங்கள் இருக்க வேண்டும்?
- உருவாக்கப்படும் ஆவணம் - சரியாகத் தான் உருவாக்கப்பட்டிருக்கிறது என்பதற்கான உறுதி என்ன?

இவற்றை ஒன்றொன்றாகப் பார்ப்போம்.

- நகைக்கடைக்காரரைப் போய் நம்முடைய மென்பொருள் நிறுவனத்தில் இருந்து யார் பார்ப்பார்கள்?

மென்பொருள் நிறுவனத்தில் இருந்து நகைக்கடைக்காரரைப் போய்ப் பார்ப்பவருக்கு

- நகைக்கடை வியாபாரம் பற்றிய விவரங்கள் தெரிந்திருக்க வேண்டும். இல்லாவிட்டால், நகைக்கடை முதலாளி சொல்லும் விவரங்கள் புரியாமல் போய் விடும்.
- அந்த விவரங்களைக் குறிப்பெடுத்து, நம்முடைய பொறியாளர்களுக்குப் புரியும் வகையில் எழுதத் தெரிந்திருக்க வேண்டும். \

இந்தத் திறமைகளைக் கொண்டவரை நம்முடைய வாடிக்கையாளரின் (நகைக்கடைக்காரரின்) வியாபாரம் பற்றி நன்கு தெரிந்தவர் என்று சொல்லலாம் தானே! இப்படிப்பட்ட திறமைகளைக் கொண்டவரைத் தான் நிறுவனங்கள் வாடிக்கையாளரைச் சந்திக்க அனுப்புகின்றன. அவர்களுக்கு 'வணிக ஆய்வாளர்' ('பிசினஸ் அனலிஸ்ட் (Business Analyst)') என்று பெயர். சில நிறுவனங்களில் இந்த வேலையை மேலாளர் நிலையில் உள்ள ஊழியர்கள் செய்வார்கள்.

- இந்த ஆவணப்படுத்தும் வேலையை வாடிக்கையாளர் செய்வாரா, வாடிக்கையாளரைப் போய்ப் பார்க்கும் நம்முடைய ஆள் செய்வாரா?

நீங்கள் கடைக்குப் போய் வரிசையாகப் பல பொருட்கள் வேண்டும் என்று ஒவ்வொன்றாகச் சொல்கிறீர்கள். நீங்கள் சொல்வதை எழுதி வைத்துக் கொள்ள வேண்டியது வாடிக்கையாளராகிய உங்கள் கடமையா? கடைக்காரரது கடமையா? கடைக்காரரது தானே! இங்கும் அதே நிலை தான்! நகைக்கடைக்காரரைப் போய்ப் பார்க்கும் நம்முடைய ஊழியர் தான், விவரங்களைத் தொகுத்து எழுதியோ ஆவணப்படுத்தியோ வைத்துக் கொள்ள வேண்டும்.

- அந்த ஆவணத்தில் என்னென்ன விவரங்கள் இருக்க வேண்டும்?

வாடிக்கையாளரது தேவைகள் அனைத்தும் ஆவணத்தில் குறிப்பிடப் பட்டிருக்க வேண்டும். நாம் பார்த்த எடுத்துக்காட்டின் படி, நகைக்கடைக்கு இணையத்தளம் வடிவமைக்கும் வேலையைப் பற்றிய முழு விவரங்கள் இடம் பெற்றிருக்க வேண்டும்.

- உருவாக்கப்படும் ஆவணம் - சரியாகத் தான் உருவாக்கப்பட்டிருக்கிறது என்பதற்கான உறுதி என்ன?

வாடிக்கையாளர் தேவை ஆவணத்தின் அடிப்படையில் தான் மொத்தத் திட்டப்பணியும் அமைந்திருக்கும். எனவே, இந்த ஆவணத்தை நிறுவனத்தில் உள்ள மூத்த ஊழியர்கள் (முதுநிலை வணிக ஆய்வாளரோ முதுநிலை மேலாளரோ) ஒரு முறைக்கு இருமுறை நன்கு ஆராய்ந்து பார்ப்பார்கள். அவர்கள் பார்த்து, ஆராய்ந்து, ஒப்புக்கொண்ட பிறகு - இந்த ஆவணம் திட்டப்பணியில் இருக்கும் ஊழியர்களுக்குக் கொடுக்கப்படும்.

சாப்ட்வேர் உருவாக்கம் எங்கு தொடங்குகிறது?

ஒரு சாப்ட்வேர் உருவாகும் முதல் இடம் - இந்த ஆவணமாக்கல் இடம் தான்! ஆக, இது வரை நாம் அறிந்து கொண்டவை

* வாடிக்கையாளர் தேவைகளை அறிய நம்முடைய ஊழியர் வாடிக்கையாளரைப் பார்த்து முழு விவரங்களையும் சேர்ப்பார்.

* சேர்த்த விவரங்களை ஆவணமாக்கி முதுநிலை ஊழியர்கள் பார்வைக்குக் கொடுப்பார்.

* அவர்கள் ஒப்புதலுக்குப் பிறகு ஆவணம் - ஊழியர்களுக்குக் கொடுக்கப்படும்.

இது தான் சாப்ட்வேர் உருவாக்கத்தின் தொடக்கப் பகுதி ஆகும். இந்த ஆவணத்தை வைத்துக் கொண்டு ஊழியர்கள் என்ன செய்வார்கள்? ஊழியர்களுக்கு அணித்தலைவராக இருப்பவர் என்ன செய்வார்? தொடர்ந்து பயணிப்போம்!

சாப்ட்வேர் டெஸ்டிங் - திட்டமிடல்

வாடிக்கையாளர் தேவைகள் பற்றிப் போதுமான விவரங்களைச் சேர்த்த பிறகு, மென்பொருள் நிறுவனம் செய்ய வேண்டிய அடுத்த வேலை - வேலையைத் திட்டமிடுவது.

திட்டமிடல் என்றால் என்ன?

எளிதான விசயம் தான்!
1) யார் யார் என்னென்ன வேலை செய்வது?
2) எப்போது செய்வது?
3) எப்படிச் செய்வது?
என்று திட்டமிடுவதைத் தான் திட்டமிடல் ('Planning') என்று சொல்கிறார்கள்.
இதில் யார் யார் என்னென்ன வேலையைச் செய்வது, என்று திட்டமிடும் போது ஊழியர் ஒவ்வொருவருக்கும் இருக்கும் திறமைகள் என்னென்ன, அவர்களுடைய பலம் என்ன, பலவீனம் என்ன, என்பதைத் தெரிந்து திட்டமிட வேண்டும். இது மட்டுமல்லாது ஊழியர்களுக்கு ஏதாவது தொழில்நுட்பப் பயிற்சி தேவைப்படுகிறதா, வேலை செய்யத் தேவையான மென்பொருட்கள் அனைத்தும் நிறுவனத்திடம் இருக்கிறதா என்பதையும் சேர்த்துத் திட்டமிட வேண்டும்.
எடுத்துக்காட்டாக, நீங்கள் 'ஆண்டிராய்டு' செயலி ஒன்றைச் சோதிக்கப் போகிறீர்கள் என்றால், அந்தச் செயலியை உருவாக்க,
1) உருவாக்குநர் (டெவலப்பர்) அணிக்குப் போதிய பயிற்சியும் அனுபவமும் உள்ளனவா?
2) டெஸ்டர்களுக்கு அனுபவமும் பயிற்சியும் உள்ளனவா?
அப்படிப் பயிற்சி தேவையெனில் எத்தனை நாட்கள் பயிற்சிக்குத் தேவைப்படும்? அதற்கான பயிற்சியாளர்களை வெளியில் இருந்து அழைத்து வர வேண்டுமா? நிறுவனத்தில் ஏற்கெனவே இருக்கும் பயிற்சியாளர்களும் பயிற்சிகளும் போதுமா?
3) ஆண்டிராய்டு செயலியைச் சோதிக்கத் தேவையான சூழல் நிறுவனத்திடம் உள்ளதா?
4) தானியங்கிச் சோதனை (ஆட்டோமேசன் டெஸ்டிங்) தேவையெனில் அதற்கான திறமையும் அனுபவமும் டெஸ்டர்களுக்கு இருக்கிறதா? ஆட்டோமேசனுக்குத் தேவையான மென்பொருட்கள் ஏற்கெனவே இருக்கின்றனவா?
5) வாடிக்கையாளர் விரும்பும் நேரத்தில், மென்பொருளை உருவாக்கிச் சோதித்து விட முடியுமா? முடியாதா? அப்படி முடியாது என்றால், இப்போதே இன்னும் சில உறுப்பினர்களை அணியில் சேர்த்துக் கொள்ளலாமா? அவர்களுக்கு யார் பயிற்சியளிப்பது? என்பன முக்கியமான கேள்விகள் ஆகும். இந்தக் கேள்விகளுக்குத் தயாராவதே திட்டமிடல் ஆகும். பொதுவாகத் திட்டமிடலில் நாம் பார்த்த எல்லாக் கேள்விகளுக்கும் சரியான பதிலை அணியின் தலைமைப் பொறுப்பில் இருக்கும் ஒருவரால் தான் சொல்ல முடியும்.

அவருக்குத் தான் அணி உறுப்பினர்கள் ஒவ்வொருவரின் பலம், பலவீனம், அணிக்குத் தேவைப்படும் பயிற்சிகள் ஆகியன பற்றிய விவரங்கள் தெரியும். எனவே, சாப்ட்வேர் டெவலப்மென்டின் இந்தக் கட்டத்தில் அணித்தலைவர் போதுமான விவரங்களைச் சேர்த்து திட்டமிடல் ஆவணத்தை உருவாக்குவார்.

மேல் சொன்ன விவரங்களுடன், இந்தத் திட்டமிடல் ஆவணத்தில் 'டெஸ்ட் ஸ்டிராடஜி' (சோதனை உத்தி) (Test Strategy) ஏதேனும் இருந்தால் அதுவும் சேர்க்கப்படும். இதென்ன புதிதாக இருக்கிறது? சோதனை திட்டம் புரிகிறது! சோதனை உத்தி என்றால் என்ன என்று கேட்கிறீர்களா? பார்த்து விடுவோம்!

சோதனை திட்டம் என்பது, யார், என்ன, எப்போது, எப்படி வேலை செய்யப் போகிறார்கள் என்று மொத்தத் திட்டப்பணிக்கும் சேர்த்துச் சொல்வது ஆகும்.

சோதனை உத்தி:

ஓர் எடுத்துக்காட்டு பார்ப்போம்! நீங்கள் மட்டைப்பந்து அணிக்குத் (கிரிக்கெட் தான்!) தலைவர் என்று வைத்துக் கொள்ளுங்கள். உங்கள் அணியில் எந்தெந்த வீரர்களுக்குப் பயிற்சி கொடுக்க வேண்டும், யார் யாருக்கு பேட்டிங் பயிற்சி, யார் யாருக்கு பௌலிங் பயிற்சி, எங்கே பயிற்சி எடுக்க வேண்டும், எப்போது எடுக்க வேண்டும் என்பன பற்றியெல்லாம் திட்டமிட்டு ஆவணத்தை உருவாக்கினால் அது திட்ட ஆவணம் ஆகிறது.

அதே நேரத்தில், எதிரணியின் முக்கியமான வீரர் ஒருவரை வீழ்த்த என்ன செய்ய வேண்டும்?, யார் பந்து வீச வேண்டும், மைதானத்தில் எந்தெந்த இடத்தில் யார் யாரை நிறுத்த வேண்டும் என்று முடிவு செய்வது பற்றிப் பேசினால் அதற்குப் பெயர் உத்தி ஆகும்.

திட்டத்திற்கும் உத்திக்கும் உள்ள வேறுபாடு புரிகிறதா? உத்தி என்பது, ஒரு குறிப்பிட்ட வேலையை எப்படிச் செய்வது என்று பார்ப்பது ஆகும்.

இங்கே நாம் சோதனை உத்தி ('டெஸ்டிங் ஸ்டிராடஜி') பற்றிப் பேசினால் திட்டப்பணியின் முக்கியமான தேவைகளை எல்லாம் எப்படிச் சோதிக்கப் போகிறோம் என்று பார்ப்பது ஆகும். திட்டப்பணியைப் பற்றி நன்கு தெரிந்து வைத்திருக்கும் அணித்தலைவரோ மூத்த உறுப்பினர்களுள் யாரோ ஒருவரோ இந்த ஆவணத்தை உருவாக்குவார்கள்.

சோதனை உத்தி ஆவணத்தில் என்னென்ன இருக்கும்?

இந்த ஆவணத்தில் உத்தியின் நோக்கம் என்ன, அதை எப்படிச் செய்யப் போகிறோம், அதற்குத் தேவைப்படும் சூழல் என்ன (சூழல் என்பது என்னென்ன மென்பொருட்கள் தேவைப்படும், என்னென்ன வன்பொருட்கள் தேவைப்படும் ஆகியன பற்றியது), உத்தியைக் கையாண்டு சோதிக்க எத்தனை நாட்கள் ஆகும் என்பது போன்ற விவரங்கள் அடங்கியிருக்கும்.

இப்போது சோதனைக்கான திட்டமும் கையில் இருக்கிறது, சோதனை உத்திக்கான வழிமுறைகளும் கையில் இருக்கின்றன. அப்படியானால் சோதிக்கத் (டெஸ்டிங்) தொடங்கி

விடலாமா? என்று கேட்கிறீர்களா? டெஸ்டிங்கைத் தொடங்குவதற்கு முன்னர் முதன்மையான ஒரு வேலை மீதியிருக்கிறது. அது என்ன? என்று அடுத்த பதிவில் பார்ப்போம்.

டெஸ்ட் கேஸ் எழுதலாம் வாங்க !

இதுவரை நாம், சாப்ட்வேர் டெஸ்டிங்கைத் திட்டமிடுவது, சோதனைக்கான உத்தி ஆவணத்தை உருவாக்குவது ஆகியவற்றைப் பார்த்து விட்டோம். இப்போது நாம் செய்யவிருப்பது - சோதனைக்கு நம்மை ஆயத்தப்படுத்துவது! இதென்ன சோதனைக்கு ஆயத்தமாகும் சோதனையா என்று யோசிக்க வேண்டாம்! எளிதானது தான்! திட்டமிடலில் தொடங்கி, உத்தி வரை நாம் டெஸ்டிங்கு நம்மையும் நம்முடைய அணியையும் ஆயத்தப்படுத்தி விட்டோம். ஆனால் முதன்மையான ஒன்று - டெஸ்டிங்கிற்குத் தேவையான மென்பொருள்! அது உருவானால் தானே நாம் சோதனையைத் தொடங்கவே முடியும்? அப்படியானால் மென்பொருள் உருவாகும் வரை (இதைத் தான் 'டெவலப்மென்ட்' என்கிறார்கள். இதை உருவாக்குபவர்கள் 'டெவலப்பர்கள்' எனப்படுவர்.) என்ன செய்வது?
எந்தெந்த வழிகளில் எல்லாம் மென்பொருளைச் சோதிக்கலாம் என ஆராயலாம். அந்த வழிகளை எல்லாம் ஆராய்ந்து பார்ப்பது ஓர் அணி அல்லவா? எனவே ஒருவர் டெஸ்டிங்கிற்குக் கண்டுபிடிக்கும் வழிகளை மற்றவர்களுக்கும் தெரியப்படுத்தும் பொருட்டு, எழுதிச் சேமித்து வைத்துக் கொள்ளலாம். இவ்வாறாக, ஒரு மென்பொருளை எப்படியெல்லாம் சோதிக்கலாம் என்பதை எழுதிச் சேமித்து வைப்பதைத் தான் 'டெஸ்ட் கேஸ்' என்கிறார்கள். (டெஸ்ட் கேசை தமிழில் எப்படிச் சொல்லலாம் என யோசிக்கிறீர்களா? 'ஆய்வு நேர்வு' எனத் தனித்தமிழில் சொல்கிறது விக்கிப்பீடியா.)

டெஸ்ட் கேஸ் எழுதலாம் வாங்க!

டெஸ்ட் கேஸ் எப்படி எழுதுவது எனப் பார்ப்போம். பொதுவாக, டெஸ்ட் கேசை, எக்செல், லிபர் கால்க் போன்ற அட்டவணைச் செயலிகளை டெஸ்ட் கேஸ் எழுதப் பயன்படுத்துவார்கள். 'டெஸ்ட் லிங்க்' (http://testlink.org/) போன்ற பயனுள்ள கட்டற்ற மென்பொருளும் டெஸ்ட் கேஸ் எழுதப் பயன்படுத்தப்படுகிறது.

டெஸ்ட் கேஸ் எழுதுவதற்கு முன்னர், வாடிக்கையாளர் தேவை ஆவணத்தில் இருப்பது போல, மொத்தத் திட்டப்பணியையும் பல்வேறு உருப்படிகளாகப் பிரித்து விடுவார்கள். ஒவ்வோர் உருப்படியையும் 'மாட்யூல்' என்று சொல்வார்கள். இந்த உருப்படிகளை ஒவ்வொரு டெஸ்டருக்கும் பிரித்துக் கொடுத்து விடுவார்கள். இப்போது டெஸ்டரின் வேலை கொடுக்கப்பட்ட உருப்படிக்கு, டெஸ்ட் கேஸ் எழுதுவது தான்!

எப்படி எழுதுவது ?

1. டெஸ்ட் கேஸ் எழுதுவதற்கு முன் - உருப்படியின் பெயரை 'டெஸ்ட் சூட்' (Test Suite) ஆக வைத்துக் கொள்ள வேண்டும்.

எ.கா. முக நூல் தான் உங்களுடைய திட்டப்பணி என்று வைத்துக் கொள்ளுங்கள். முக நூலில் பயனர் உள்நுழையும் பக்கத்தைச் சோதிப்பது தான் உங்கள் உருப்படி என்றால், இங்கு, பயனர் உள்நுழைவைச் சோதிப்பது தான் உங்களுடைய டெஸ்ட் சூட் ஆகும். (இதை 'டெஸ்ட் சினரியோ' (Test Scenario) என்றும் சொல்வதுண்டு.) இந்த டெஸ்ட் சூட்டில்,

1) சரியான பயனர் பெயர், தவறான கடவுச்சொல் கொடுத்தால் என்ன ஆகும்?
2) சரியான பயனர் பெயர், சரியான கடவுச்சொல் கொடுத்தால் என்ன ஆகும்?
3) தவறான பயனர் பெயர், தவறான கடவுச்சொல் கொடுத்தால் என்ன ஆகும்?

ஆகிய மூன்றையும் நாம் சோதிக்க வேண்டும். இவை ஒவ்வொன்றும் ஒரு டெஸ்ட் கேஸ் ஆகக் கருதப்படும்.

எனவே,

- டெஸ்ட் சூட் என்பது பயனர் உள் நுழைவுச் சோதனை
- டெஸ்ட் கேஸ் என்பது டெஸ்ட் சூட்டில் உள்ள ஒவ்வொரு வழியையும் குறிப்பது.

கீழ் உள்ள டெஸ்ட் கேசை எடுத்துக்காட்டுக்குப் பார்க்கலாம். இங்கு எதிர்பார்க்கும் முடிவும் உண்மையான முடிவும் பொருந்தியிருக்க வேண்டும். அப்படிப் பொருந்தினால் நாம் செய்யும் சோதனை வெற்றி எனக் கொள்ளலாம். உண்மையான முடிவை டெஸ்ட் கேஸ் எழுதும் போதே எழுதிவிட முடியாது. மென்பொருள் உருவாக்கப்பட்டு வந்தவுடன் சோதிக்கும் போது அதை நிரப்புவார்கள்.

டெஸ்ட் கேஸ் எண்		டெஸ்ட் கேஸ்	முன் செய்யவேண்டியவை (ஏதும் இருந்தால்)	சோதனை உள்ளீடு	படிகள்	எதிர்பார்க்கும் முடிவு	உண்மையான முடிவு	தேர்ச்சி
1		சரியான பயனர் உள்நுழைய முடிகிறதா எனப் பார்.	இல்லை	பயனர்: அ ஆ இ கடவுச்சொல்: க்ச்ட்	1. பயனர் பெயரை உள்ளிடு 2. கடவுச்சொல்லை உள்ளிடு	பயனர் நுழைய முடியும்		ஆம்/ இல்லை
2		சரியான பயனர்	இல்லை	பயனர்: அ	1. பயனர் பெயரை	பயனர் நுழைய		ஆம்/

		தவறான கடவுச்சொல்லுடன் உள்நுழைய முடிகிறதா எனப் பார்.		ஆ இ கடவுச்சொல்: க்ச்	உள்ளிடு 2. கடவுச்சொல்லை உள்ளிடு	முடியாது		இல்லைம்
3		தவறான பயனர் உள்நுழைய முடிகிறதா எனப் பார்.	இல்லை	பயனர்: அ ஆ கடவுச்சொல்: க்ச்ட்	1. பயனர் பெயரை உள்ளிடு 2. கடவுச்சொல்லை உள்ளிடு	பயனர் நுழைய முடியாது		ஆம்/ இல்லை

எடுத்துக்காட்டுக்கு:

இப்படி டெஸ்ட் கேஸ் எழுதி மென்பொருளுக்காக ஆயத்த நிலையில் இருக்க வேண்டும். மென்பொருளை உருவாக்கி முடித்தவுடன் அதில் நாம் எழுதியுள்ள டெஸ்ட் கேஸ்களைப் பொருத்திச் சோதித்துப் பார்க்க வேண்டும். அது சரி! நாம் வாடிக்கையாளர் தேவை ஆவணத்தில் உள்ள எல்லாத் தேவைகளுக்கும் டெஸ்ட் கேஸ்களை எழுதி விட்டோமா என்று எப்படிப் பார்ப்பது? தொடர்ந்து பார்ப்போம்!

தேவை சுவட்டு ஆவணம் என்றால் என்ன?

உருவாக்குநர்கள் (டெவலப்பர்கள்) மென்பொருளை உருவாக்கிக் கொண்டிருக்கும் நேரத்தில் டெஸ்டர்கள் உருவாக்கப்பட்டுக் கொண்டிருக்கும் மென்பொருளை எந்தெந்த வழிகளில் எல்லாம் சோதிக்கலாம் என்பதை எழுதி வைக்கிறார்கள். இதைத்தான் நாம் டெஸ்ட் கேஸ் என்று முந்தைய பதிவில் பார்த்தோம். இந்த டெஸ்ட் கேஸ்களை எழுதுவதற்கு வாடிக்கையாளர் தேவை ஆவணத்தை அடிப்படையாக எடுத்துக் கொள்வார்கள் என்பதையும் பார்த்து விட்டோம்.

உருவாக்குநர்கள் மென்பொருளை உருவாக்கி முடித்ததும் டெஸ்டர்கள் ஏற்கெனவே எழுதி வைத்திருக்கும் டெஸ்ட் கேஸ்களின் துணை கொண்டு சோதிப்பதைத் தொடங்கலாம். ஆனால் அப்படிச் சோதனையைத் தொடங்குவதற்கு முன்னர், டெஸ்ட் கேஸ்கள் முழுமையாக எழுதி முடிக்கப்பட்டு விட்டனவா எனப் பார்க்க வேண்டும் அல்லவா? அதை எப்படிப் பார்ப்பது? இங்குத் தான் டெஸ்ட் கேஸ்களையும் வாடிக்கையாளர் தேவை ஆவணத்தையும் ஒப்பிடும் 'தேவை சுவட்டு ஆவணம்' ('Requirements Traceability Matrix – ரெக்குயர்மென்ட்ஸ் டிரேசபிளிட்டி மேட்ரிக்ஸ்') தேவைப்படுகிறது.

தேவை சுவட்டு ஆவணம் என்றால் என்ன?

இதென்ன பெயர் - தேவை சுவட்டு ஆவணம் - ஒன்றுமே புரியும்படி இல்லை என்கிறீர்களா? கவலை வேண்டாம்! விளக்கமாகப் பார்த்து விடலாம். அதைப் பார்ப்பதற்கு முன் பழைய பதிவுகளில் இருந்து உங்களிடம் ஒரு கேள்வி - வாடிக்கையாளருடைய தேவைகள் அனைத்தையும் நாம் எங்கே பார்க்கலாம்? சரியாகச் சொன்னீர்கள் - 'வாடிக்கையாளர் தேவை ஆவணத்தில்' இருந்து தான்! இந்த வா. தே. ஆவணத்தையும் டெஸ்டர்கள் எழுதி வைத்திருக்கும் டெஸ்ட் கேஸ்களையும் ஒப்பிட்டு ஓர் ஆவணத்தை உருவாக்கி விட்டால், அந்த ஆவணத்திற்குப் பெயர் தான் தேவை சுவட்டு ஆவணம் ஆகும். இந்தத் தேவை சுவட்டு ஆவணத்தில் எல்லாத் தேவைகளையும் அவற்றிற்கு நேர் எதிராக டெஸ்ட் கேஸ்களையும் பட்டியலிட்டுச் சரிபார்ப்பார்கள்.

எ.கா:

தேவை	வா.தே.ஆ 1.1	வா.தே.ஆ 1.2	வா.தே.ஆ 1.3	வா.தே.ஆ 2.1	வா.தே.ஆ 2.2	வா.தே.ஆ 3.1

டெஸ்ட் கேஸ்						
1.1.1	x					
1.1.2		x				
1.1.3		x	x			
1.1.4		x	x	x		x
1.2.1		x	x		x	
1.2.2		x	X	X	X	

தேவை சுவட்டு ஆவணம் - வகைகள் :

1) நேர்நிலை சுவட்டு ஆவணம்:

வாடிக்கையாளர் தேவைகளை டெஸ்ட் கேஸ்களுடன் பொருத்திப் பார்த்து உருவாக்குவது நேர்நிலை சுவட்டு ஆவணம் ('பார்வர்டு டிரேசபிளிட்டி - Forward Traceability') எனப்படுகிறது. இங்கு முதன்மை நோக்கம் சரியான திசையில் திட்டப்பணி செல்கிறதா, தேவையான மென்பொருளைச் சரியாக உருவாக்குகிறோமா, அதற்கான டெஸ்ட்கேஸ்கள் இருக்கின்றனவா என்று பார்ப்பதாகும்.

2) எதிர்நிலை சுவட்டு ஆவணம்:

எதிர்நிலையில் டெஸ்ட் கேஸ்களை வைத்துக் கொண்டு வாடிக்கையாளர் தேவைகளைத் திட்டப்பணி (அல்லது மென்பொருள்) நிறைவு செய்கிறதா என்று பார்ப்பது! அதாவது வாடிக்கையாளர் தேவை ஆவணத்தைக் கொண்டு டெஸ்ட் கேஸ்களை எழுதி, அந்த டெஸ்ட் கேஸ்களை மட்டுமே மென்பொருள் கொண்டிருக்கிறதா அல்லது வாடிக்கையாளருக்குத் தேவையில்லாதவற்றையும் சேர்த்து மென்பொருளில் டெவலப்பர்கள் (தெரிந்தோ தெரியாமலோ) வைத்து விட்டார்களா என்று பார்ப்பது தான் எதிர்நிலை சுவட்டு ஆவணம் ஆகும்.

யார் உருவாக்குவார்கள்?

ஓர் அணியின் தலைவருக்குத் தான் அணி உறுப்பினர்கள் ஒவ்வொருவரும் எத்தனை டெஸ்ட் கேஸ்கள் எழுதியிருக்கிறார்கள் என்பதும் அவை எந்தெந்த உருப்படிகளைச் சோதிக்க உதவும் என்பது முழுமையாகத் தெரியும். எனவே இந்த தேவை சுவட்டு ஆவணத்தை உருவாக்குவதற்குத் தகுதியான ஆள் அணித் தலைவர் தாம்! சில நிறுவனங்களில் அணியில் உள்ள மூத்த உறுப்பினர்களும் ஆவண உருவாக்கத்தில் ஈடுபடுவது உண்டு.

நன்மைகள்:

1) ஒன்று விடாமல் வாடிக்கையாளரின் எல்லாத் தேவைகளையும் சோதிக்கத் தேவையான டெஸ்ட் கேஸ்களை எழுதிவிட்டோமா என்று அறிந்து கொள்ளலாம்.
2) வாடிக்கையாளரின் தேவைகளில் ஏதேனும் குழப்பங்களோ சந்தேகங்களோ இருந்தால் அவற்றைத் தெரிந்து திருத்திக் கொள்ள இந்த ஆவணம் உதவும்.

வேறென்ன செய்யலாம்:

மென்பொருள் உருவாக்கம் ஒரு பக்கம் போய்க் கொண்டிருக்கிறது. அந்நேரத்தில் தான் டெஸ்டர்கள் இது போல பல ஆவணங்களை உருவாக்கிச் சோதனைக்குத் தங்களை ஆயத்தப்படுத்திக் கொள்வார்கள். ஆவணங்கள் எல்லாம் முடிந்த பிறகு, டெஸ்டிங் செய்யத் தேவைப்படும் சூழலை உருவாக்குவதில் ஈடுபடுவார்கள்.

சோதனைக்கு உகந்த சூழல்:

- நாம் பயர்பாக்ஸ் அலைபேசி ஒன்றைச் சோதிக்கிறோம் என்றால், பயர்பாக்ஸ் அலைபேசி ஒன்றையோ அல்லது பயர்பாக்ஸ் நிறுவப்பட்டுள்ள கணினி ஒன்றையோ ஆயத்தப்படுத்தி வைப்பது
- இணையத்தளம் ஒன்றைச் சோதிக்கிறோம் என்றால் தேவைப்படும் உலாவிகளை (பயர்பாக்ஸ், குரோம்) நிறுவி வைத்துக் கொள்வது
- இதைப் போல தேவைப்படும் பிற மென்பொருட்களை நிறுவிக் கொள்வது

ஆகியவற்றைத் தான் சோதனைக்கு உகந்த சூழலை உருவாக்குவது என்கிறோம்.
ஒருவழியாக, டெஸ்ட் கேஸ் எழுதி, சூழலை உருவாக்கிச் சோதனையைத் தொடங்கும் புள்ளிக்கு வந்து விட்டோம். டெவலப்பர்கள் மென்பொருளைக் கொடுத்த உடன், சோதனையைத் தொடங்கி விடலாம். நம்மிடம் மென்பொருளைக் கொடுத்துச் சோதிக்கச் சொல்வதற்கு முன், டெவலப்பர்களே ஒரு முறை சோதித்துப் பார்ப்பார்கள்.
'டெவலப்பர்கள் மென்பொருளைச் சோதிப்பார்களா? சோதித்துப் பார்ப்பதற்குத் தானே நாம் இருக்கிறோம்! பிறகு அவர்கள் என்ன சோதிப்பார்கள்?' என்கிறீர்களா? அதைப் பற்றிப் பேசுவோம் - அடுத்த பதிவில்!

மென்பொருள் உருவாக்கமும் சோதனையும்

டெஸ்டர்கள் மென்பொருள் சோதனைக்கு ஆயத்தமாகிக் கொண்டிருந்த வேளையில் (அதாவது, டெஸ்டர்கள் டெஸ்ட் கேஸ் எழுதிய போதும் அதற்கு முன்பும்) உருவாக்குநர்கள் (டெவலப்பர்கள்) என்ன செய்து கொண்டு இருந்திருப்பார்கள் என்று உங்களால் ஊகிக்க முடிகிறதா? சரியாகச் சொன்னீர்கள் - மென்பொருளை உருவாக்கிக் கொண்டிருப்பார்கள். அவர்கள் வேலையே அது தானே! ஆனால் மென்பொருளை உருவாக்குவதோடு உருவாக்குநர்களின் வேலை முடிந்து விடுவதில்லை! மென்பொருளை முதல் நிலையில் சோதிப்பதும் அவர்கள் வேலை தான்!
என்ன குழப்புகிறீர்கள்? மென்பொருளை உருவாக்குவதால் 'உருவாக்குநர்கள் (டெவலப்பர்கள்)' என்கிறீர்கள். பிறகு, சோதிப்பதும் அவர்கள் தாம் என்று சொல்லிச் சோதிக்கிறீர்களே என்கிறீர்களா? குழப்பம் வேண்டாம்! மென்பொருள் இல்லாமல் வேறு ஏதாவது ஓர் எடுத்துக்காட்டு மூலம் இதைப் பார்ப்போம் - புரிந்து விடும்.

நண்பருக்கு விருந்தும் அம்மாவின் சோதனையும்:

நண்பர் ஒருவரை வீட்டிற்கு விருந்திற்குக் கூப்பிட்டிருக்கிறோம் என்று வைத்துக் கொள்ளுங்கள். நம்முடைய அம்மா விருந்தைச் சமைக்கிறார்கள்; இங்கு, நண்பர் தாம் வாடிக்கையாளர், அம்மா தான் உருவாக்குநர். விருந்தில், சோறு, குழம்பு, கூட்டு, பொரியல், வடை, அப்பளம், பாயாசம் எனப் பல உணவுவகைகளை அம்மா உருவாக்க வேண்டும். இப்போது அம்மா - குழம்பு வைத்து முடித்து விட்டார்கள். குழம்பு நன்றாகக் கொதி வந்து அடுப்பில் இருந்து இறக்கி வைக்கும்போது, அம்மா இரண்டு சொட்டு கரண்டியில் எடுத்துச் சுவைத்துப் பார்ப்பார்கள் அல்லவா - உப்பு, உறைப்பு எல்லாம் சரியாக இருக்கிறதா என்று! இது தான் முதல் நிலை சோதனை! அதாவது உருவாக்குநரே தாம் உருவாக்கியுள்ள பொருள் சரியாக இருக்கிறதா என்று சோதித்துப் பார்ப்பது! இங்கு அம்மா எல்லா வகை உணவையும் சமைத்த பின்னர் தனித்தனியாகச் சோதித்துப் பார்ப்பார்கள்! இப்படி ஒவ்வோர் உருப்படியாகச் சோதிப்பதைத் தான் 'தனி உருப்படிச் சோதனை' என்றோ 'அலகுச் சோதனை' என்றோ சொல்கிறார்கள். ஆங்கிலத்தில் இச்சோதனைக்கு 'யூனிட் டெஸ்டிங்' என்று பெயர்.

இச்சோதனையைச் செய்ய, ஜேயூனிட் (http://junit.org/) போன்ற கட்டற்ற வடிவமைப்பு மென்பொருட்கள் பயன்படுத்தப்படுகின்றன. இப்படியாக, அலகுச் சோதனையை முடித்த பிறகு ஒருங்கிணைப்புச் சோதனையைச் செய்வார்கள். அதென்ன ஒருங்கிணைப்புச் சோதனை? பார்த்து விடுவோம்!

இரு சக்கர வண்டியும் ஒருங்கிணைப்புச் சோதனையும்:

ஒரு பெரிய இருசக்கர வண்டிகள் தயாரிக்கும் நிறுவனம். அந்நிறுவனத்தின் ஒவ்வோரு துறையிலும் ஒவ்வொரு பாகத்தைத் தயாரிக்கிறார்கள். ஒரு துறையில் சக்கரங்கள், ஒரு துறையில் வேகமானி (ஸ்பீடோமீட்டர்), மற்றொரு துறையில் இருக்கைகள் இப்படி! நாம் மேலே பார்த்தது போல, ஒவ்வொரு துறை வல்லுநர்களும் தத்தம் உருவாக்கிய பாகத்தை 'யூனிட் டெஸ்டிங்' செய்வார்கள்.

யூனிட் டெஸ்டிங் என்பது தனித்தனியே ஒவ்வொரு பாகமும் சரியாக இயங்குகிறதா? என்று சோதிப்பதாகும். சில அல்லது பல சமயங்களில் ஒவ்வொரு பாகமும் தனித்தனியே சரியாக இயங்கும்; அவற்றை மற்ற பாகங்களுடன் சேர்க்கும் போது சரிவர இயங்காமல் போகும். அதாவது, சக்கரம் தனியாக எந்தவிதச் சிக்கலும் இல்லாமல் சுழலும்; வேகமானி தனியாக சரியாக இயங்கும். ஆனால் இவை இரண்டையும் இணைக்கும் போது, சக்கரத்தின் வேகத்திற்கேற்ப வேகமானியின் முள் இருக்க வேண்டும். இந்த இணைப்பு சரியாக இருக்கிறதா என்று பார்ப்பது தான் ஒருங்கிணைப்புச் சோதனையாகும். இதை ஆங்கிலத்தில் 'இன்டக்ரேஷன் டெஸ்டிங்' (Intergration Testing) என்று சொல்வார்கள்.

- ஒருங்கிணைப்புச் சோதனையை யார் செய்வார்கள்?

ஒருங்கிணைப்புச் சோதனை மற்ற சோதனைகளைப் போல டெஸ்டர்களால் செய்யப்படும் ஒரு சோதனையாகும்.

- ஒருங்கிணைப்புச் சோதனையை ஏன் உருவாக்குநர்களே செய்யக்கூடாது?

நல்ல கேள்வி! பொது நிலையில், உருவாக்குநர்களுக்குத் தத்தம் உருவாக்கிய பாகத்தைப் பற்றிய அறிவும் பட்டறிவும் இருக்குமே தவிர, பிற பாகங்களைப் பற்றி அவர்கள் தெரிந்து வைத்திருப்பார்கள் என்று சொல்வது சரியாக அமையாது. சக்கரத்தை உருவாக்குபவருக்கு வேகமானி பற்றிய அறிவும் பட்டறிவும் இருக்க வேண்டும் என்று எதிர்பார்ப்பது நடைமுறைக்கு ஒத்து வராது. அதே நேரத்தில் டெஸ்டர்களுக்கு இவை இரண்டும் இணைந்து இப்படித்தான் வேலை செய்யும் என்பது பற்றிய தெளிவு ஓரளவு இருக்கும் என்பதால் ஒருங்கிணைப்புச் சோதனையை டெஸ்டர்கள் செய்வது தான் சரியாக இருக்கும்.

- இரு வேறு பாகங்களை இணைக்கும் போது, சில நேரங்களில் ஒரு பாகம் மட்டும் சோதனைக்குத் தயாராகி, மற்றொரு பாகம் தயாராகவில்லை எனில் என்ன செய்வார்கள்? அதாவது, வேகமானி கையில் தயாராக இருக்கிறது, ஆனால் இன்னும் சக்கரம் வந்து சேரவில்லை என்று வைத்துக் கொள்ளுங்கள். அந்த நேரத்தில் ஒருங்கிணைப்புச் சோதனையைச் செய்வது எப்படி?

இது போன்ற நேரத்தில் எந்த பாகம் இன்னும் உருவாகவில்லையோ அந்தப் பாகத்தைப் போலவே ஒரு போலி மென்பொருளைத் தற்காலிகமாக உருவாக்குவார்கள். அந்தப் போலி மென்பொருளை வைத்து சோதனையை நடத்துவார்கள். ஓர் எளிய எடுத்துக்காட்டு மூலம் இதை விளக்கலாம்.

நீங்கள் ஒரு மாணவன் கல்லூரியில் எட்டுப் பருவங்களிலும் எடுத்த மதிப்பெண்களைக் கூட்டிச் சொல்வதற்குரிய ஒரு மென்பொருளைச் (மென்பொருள் 'அ') சோதிக்கிறீர்கள் என்று வைத்துக் கொள்வோம். இந்த மென்பொருளுக்கு இன்னொரு மென்பொருள் ('ஆ') அதே மாணவன் ஒவ்வொரு பருவத்திலும் எடுத்த மதிப்பெண்களைக் கூட்டிச் சொல்லும்

வேலையைச் செய்கிறது. ஆக, மென்பொருள் 'ஆ' உருவான பிறகு தான் மென்பொருள் 'அ' வை நம்மால் சோதிக்க முடியும்.

ஸ்டப் ('Stub')

ஆனால், விதிவசத்தில் மென்பொருள் 'அ' முன்னதாகவே உருவாக்கப்பட்டு விடுகிறது. இப்போது சோதனையைச் செய்ய (அதாவது மென்பொருள் 'அ'விற்கு மதிப்பெண்களைக் கொடுக்க) மென்பொருள் 'ஆ' நமக்கு வேண்டும் - ஆனால் கையில் இல்லை. இந்த நிலையில் தான் மென்பொருள் 'ஆ' வைப் போலவே ஒரு தற்காலிகப் போலி மென்பொருளை உருவாக்கிச் சோதனையைச் செய்வார்கள். இந்த இடத்தில் மென்பொருள் 'ஆ'வை நம்பித் தான் மென்பொருள் 'அ' இருக்கிறது. மென்பொருள் 'ஆ'விற்காக உருவாக்கப்படும் தற்காலிகப் போலி மென்பொருளை ஆங்கிலத்தில் 'ஸ்டப்' (Stub) என்று சொல்வார்கள்.

டிரைவர்('Driver')

மேல் சொன்ன அதே எடுத்துக்காட்டில், மென்பொருள் 'ஆ' ஆயத்த நிலையில் இருந்து மென்பொருள் 'அ' இன்னும் உருவாக்கப்படவில்லை என்று வைத்துக் கொள்ளுங்கள். அப்போது மென்பொருள் 'அ'விற்கு மாற்றாகத் தற்காலிகப் போலி மென்பொருள் ஒன்றை உருவாக்குவார்கள். அதை 'டிரைவர்' என்று சொல்வார்கள்.

போலி என்றவுடன் தவறான எண்ணத்தில் பார்த்து விடாதீர்கள். உண்மையான மென்பொருளைப் போலவே இருப்பதால் அதைப் போலி என்கிறோம். அவ்வளவுதான்!

- சரி, இந்த 'ஸ்டப்', 'டிரைவர்' ஆகியவற்றைத் 'தற்காலிக' மென்பொருள் என்று சொல்கிறீர்களே! ஏன்?

உண்மையான மென்பொருளுக்கு மாற்றாக, அந்தந்த நேரத்தில் சோதனையை(டெஸ்டிங்கை) நடத்திப் பார்க்க வேண்டும் என்பதற்காக உருவாக்கப்படும் ஸ்டப், டிரைவர் ஆகிய போலி மென்பொருட்கள் உண்மையான மென்பொருட்கள் உருவாக்கப்பட்டவுடன் அழிக்கப்பட்டு விடும். அவற்றின் தேவை சோதனையை நடத்திவிட வேண்டும் என்பதற்கான தற்காலிகத் தேவையே! எனவே தான் அவற்றைத் தற்காலிக மென்பொருள் என்று சொல்கிறோம்.

இதுவரை அலகுச் சோதனையைப் பற்றியும், ஒருங்கிணைப்புச் சோதனையைப் பற்றியும் ஓரளவு பார்த்து விட்டோம். இனிமேல் தான் முழுமையான சோதனையைப் பார்க்கப் போகிறோம். அடுத்த பதிவில் அதைப் பற்றிப் பேசுவோம்.

சோதிக்கத் தொடங்குவோம்

அலகு(தனி உருப்படி)ச் சோதனையை உருவாக்குநர் முடித்து, ஒருங்கிணைப்புச் சோதனையை டெஸ்டர்கள் முடித்திருக்கிறார்கள். ஒவ்வோர் உருப்படியையும் உருவாக்கி அந்த உருப்படிகளை மற்ற உருப்படிகளுடன் சரிவர இணைந்து இயங்குகின்றனவா என்று இது வரை பார்த்திருக்கிறோம். ஜிமெயில், யாஹூ மெயில் போல, மின்னஞ்சல் சேவை கொடுக்கும் மென்பொருள் ஒன்றை நம்முடைய நிறுவனம் உருவாக்கிக் கொண்டிருக்கிறது என்று வைத்துக் கொள்ளுங்கள். இங்கு,

- மின்னஞ்சல் நுழைவுப் பக்கம் (லாகின் பக்கம்)
- உள்பெட்டி (இன்பாக்ஸ்)
- வெளிப்பெட்டி (சென்ட் ஐடெம்)
- தொடர்புகள்

என ஒவ்வோர் உருப்படியையும் மற்ற உருப்படிகளுடன் இணைத்துச் சோதிக்க வேண்டும். எடுத்துக்காட்டாக, நுழைவுப்பக்கத்தில் எந்தப் பெயர், கடவுச்சொல் கொடுக்கிறோமோ - அந்தத் தகவலுக்கேற்ற உள்பெட்டியைக் காட்ட வேண்டும். குப்பனின் பெயரையும் கடவுச்சொல்லையும் கொடுத்தால் சுப்பனின் உள்பெட்டியைக் காட்டக் கூடாது. இதைத் தான் ஒருங்கிணைப்புச் சோதனையில் சரிபார்க்கிறோம்.

ஆனால் ஒருங்கிணைப்புச் சோதனை மட்டுமே முழுமையான சோதனையாகி விடாது அல்லவா? எனவே, மொத்த மென்பொருளையும் ஒரு முறை முழுமையாகச் சோதிக்க வேண்டும். அதைத் தான் முழுமைச் சோதனை என்கிறார்கள். (ஆங்கிலத்தில் இதை 'ரெக்ரெசன் டெஸ்டிங்' (Regression Testing) என்பார்கள்.)

உருவாக்குநர்கள் அனைவரும் இணைந்து மொத்த மென்பொருளையும் உருவாக்கிய பிறகு, மென்பொருள் முழுமையையும் மொத்தமாகச் சோதிக்க வேண்டும். இந்தச் சோதனை, உண்மையான ஒரு வாடிக்கையாளர் மென்பொருளை எப்படிக் கையாள்வாரோ, அதே அடிப்படையில் இருக்க வேண்டும். அதாவது, மேல் சொன்ன மின்னஞ்சல் நுழைவுப்பக்கத்தை எடுத்துக்காட்டுக்கு எடுத்துக்கொள்ளுங்கள்.

பயனர்கள்,

- முதல் முறையிலேயே சரியான பயனர் பெயர், கடவுச்சொல் கொடுத்து நுழையலாம்
- முதல் முறையில் சரியான பயனர் பெயரையும் தவறான கடவுச்சொல்லையும் கொடுக்கலாம்
- முதல் முறை - தவறான பயனர் பெயரை உள்ளிடலாம்.
- பயனர் பெயர், கடவுச்சொல் - இரண்டையுமே பயனர் தவறாகக் கொடுக்கலாம்.

இப்படியாக ஒவ்வொரு பயனரும் ஒவ்வொரு விதமாகச் செயல்படலாம். ஆனால், பயனர் எப்படிச் செயல்பட்டாலும், அந்தச் செயல்பாட்டிற்குப் பொருத்தமான பதிலை மென்பொருள் கொடுக்க வேண்டும். அதாவது,

- சரியான பயனர் பெயர், கடவுச்சொல்லிற்கு, நுழைவிற்கு அடுத்த இன்பாக்ஸ் பக்கத்தைக் காட்டுவது
- தவறான கடவுச்சொல்லிற்கோ, தவறானபயனர் பெயருக்கோ 'பயனர் பெயர் / கடவுச்சொல் தவறு' எனச் செய்தி காட்டுவது

என்பன போன்ற பொருத்தமான பதில்களை கொடுக்க வேண்டும்.

ஒரு டெஸ்டராக, மேல் சொன்ன எல்லாவகைகளிலும் உருவாக்கப்பட்டுள்ள மென்பொருள் சரியாக இயங்குகிறதா என்பதைச் சோதிக்க வேண்டும். ஐயயோ! எல்லா வகைகளையும் எப்படி நினைவில் வைத்திருப்பது? ஏதாவது ஒரு வகையை மறந்து விட்டால் என்ன செய்வது? என்று கவலைப்படுகிறீர்களா? கவலையே வேண்டாம்! இந்தக் கவலையைப் போக்குவதற்குத் தான் முன்னரே எல்லாவகைகளையும் உள்ளடக்கி டெஸ்ட் கேஸ்களை எழுதி வைத்திருக்கிறோம். (பார்க்க: http://www.kaniyam.com/software-testing-8-write-test-case/)

இப்போது நம்முடைய வேலை, அந்த டெஸ்ட் கேஸ்களை ஒவ்வொன்றாக எடுத்து மென்பொருள் மீது செயல்படுத்திப் பார்ப்பது தான்! இப்படிச் செயல்படுத்துவதில் அணியில் யார் யார் எந்தெந்த டெஸ்ட் கேஸ்களைச் செயல்படுத்த வேண்டும் என்பதை டெஸ்டிங் அணித்தலைவர் தீர்மானித்துப் பிரித்துக் கொடுப்பார். இப்படி டெஸ்ட் கேஸ்களைச் செயல்படுத்தி, வரும் முடிவுகளை 'உண்மையான முடிவு' என்பதில் பதிந்து வைக்க வேண்டும். எதிர்பார்க்கும் முடிவுகளும் உண்மையான முடிவுகளும் பொருந்தி வரும் டெஸ்ட் கேஸ்களை தேறிய டெஸ்ட் கேஸ்களாகவும், பொருந்தாத டெஸ்ட் கேஸ்களைத் தோல்வியடைந்த அல்லது தேறாத டெஸ்ட் கேஸ்களாகவும் எடுத்துக் கொள்ளலாம். இப்படித் தேறாத டெஸ்ட் கேஸ்கள் எல்லாம் நமக்கு உணர்த்தும் செய்தி ஒன்று தான் - வாடிக்கையாளர் எதிர்பார்க்கும் முடிவை நம்முடைய உருவாக்குநர்கள் உருவாக்கியிருக்கும் மென்பொருள் கொடுக்கவில்லை என்பது தான்!

இப்படித் தேறாத டெஸ்ட் கேஸ்களை உருவாக்குநர்களிடம் முறைப்படி அவ்வப்போது தெரியப்படுத்தி விட வேண்டும். அப்போது தான், அவர்கள் உடனுக்குடன் உட்கார்ந்து தவறுகளைத் திருத்தி மீண்டும் சோதிப்பதற்கு நம்மிடம் கொடுப்பார்கள்.

- தேறாத டெஸ்ட் கேஸ்களை உருவாக்குநர்களிடம் கொடுக்க வேண்டும் என்றால், எந்த உருவாக்குநர் மென்பொருளின் எந்தப் பகுதியை உருவாக்கினார் என்பது தெரிய வேண்டுமே! அதைத் தெரிந்து கொள்ள முடியுமா?
- அப்படித் தெரிந்து கொண்டாலும் ஒருவேளை அந்த உருவாக்குநர் விடுப்பில் இருந்தாலோ, வேறு திட்டப்பணிக்கு மாற்றப்பட்டிருந்தாலோ, வேறு நிறுவனத்திற்குப் போய் விட்டாலோ தேறாத பகுதிகளை எப்படித் தெரிவிப்பது?
- உருவாக்குநர்களிடம் தெரிவிப்பது என்றால் எப்படி - சொல்வது மூலமா? அல்லது மின்னஞ்சல் அனுப்ப வேண்டுமா? இல்லை மென்பொருளில் உள்ள தவறுகளைத் தெரிவிப்பதற்கென்று தனியாக மென்பொருட்கள் உள்ளனவா?
- இதையெல்லாம் தெரிந்து கொண்டு உருவாக்குநரிடம் தெரிவிக்கிறோம் என்று வைத்துக் கொள்வோம். அவர் தவற்றைத் திருத்தி யாரிடம் கொடுப்பார்? தவற்றைச் சுட்டிக்காட்டிய டெஸ்டரிமா? ஒருவேளை அந்தக் குறிப்பிட்ட டெஸ்டர் விடுப்பில்

இருந்தாலோ, வேறு திட்டப்பணிக்கு மாற்றப்பட்டிருந்தாலோ உருவாக்குநர் யாரிடம் தெரிவிப்பார்?

இப்படிப் பல கேள்விகள் இப்போது எழலாம். இந்தக் கேள்விகளுக்கெல்லாம் விடை தெரிய வேண்டும் என்றால், 'பிழை வாழ்க்கை வட்டம்' என்றால் என்ன என்று தெரிய வேண்டும். அந்த வட்டத்தைச் சுற்றுவோம் அடுத்த பதிவில்!

டெஸ்ட் கேஸ் உத்திகள் - 1

தோழர்,

அடுத்த பதிவில் 'பிழை வாழ்க்கை வட்டம்' பற்றிப் பார்ப்போம் என்று சென்ற பதிவில் சொல்லியிருக்கிறீர்கள். ஆனால், அதற்கு முன் எனக்கு ஒரு சந்தேகம் - டெஸ்ட் கேஸ் எழுதுவது பற்றிப் (http://www.kaniyam.com/software-testing-8-write-test-case/) படித்து விட்டு, ஜிமெயிலில் பயனர் உருவாக்கும் பக்கத்திற்கு டெஸ்ட் கேஸ்களை எழுதலாம் என நினைத்து ஆர்வத்தில் டெஸ்ட் கேஸ் எழுதத் தொடங்கினேன். இன்னும் முடிக்க முடியவில்லை. எழுத எழுத டெஸ்ட் கேஸ்கள் வந்து கொண்டே இருக்கின்றன; நானும் பக்கம் பக்கமாக டெஸ்ட் கேஸ்கள் எழுதிவிட்டேன். இப்படித்தான் நிறுவனங்களிலும் திட்டப்பணியில் செய்வார்களா? இல்லை டெஸ்ட் கேஸ்கள் எழுதுவதற்கு ஏதாவது எளிய வழிமுறைகள் இருக்கின்றனவா? சொல்லுங்களேன்.

இப்படிக்கு,

முகிலன்.

தோழர் முகிலனுக்கு,

வணக்கம். நல்ல கேள்வியைக் கேட்டிருக்கிறீர்கள். டெஸ்ட் கேஸ்கள் எழுதுவதற்கு எளிய பல வழிமுறைகள் இருக்கின்றன. அவற்றை முதலில் இங்கு விளக்கி விடுகிறேன். பிறகு, நாம் 'பிழை வாழ்க்கை வட்டம்' பற்றிப் பார்க்கலாம்.

டெஸ்ட் கேஸ் எழுத எளிய வழிமுறைகள் :

டெஸ்ட் கேஸ் என்பது, பொதுவாக, பயனர் எந்தெந்த விதமான உள்ளீடுகளைக் கொடுப்பார் என்பதை உணர்ந்து குறிப்பெடுத்து வைத்துக் கொள்வது தான்! ஆனாலும் எல்லாச் சூழல்களிலும் எல்லா விதமான உள்ளீடுகளையும் கொடுத்துச் சோதிப்பது என்பது இயலாத காரியம். அதே சமயம், எல்லா வகைகளிலும் சோதிப்பது என்பது இன்றியமையாதது. குழப்புகிறதா? விரிவாகவே பார்ப்போம்.

டெஸ்ட் கேஸ் எழுதுவதற்கு உத்திகள் ஏன் தேவை?

மட்டைப்பந்துப் போட்டி நடந்து கொண்டிருக்கிறது. சச்சின் டெண்டுல்கர் மட்டையோடு நின்று கொண்டிருக்கிறார். எதிரணியில் இருக்கும் அனைவரும் கூடிப் பந்து வீச்சாளரிடம் பேசி, சச்சினை வீழ்த்த உத்திகள் வகுக்கிறார்கள். அதே சமயம், மட்டையோடு நிற்பவர் பதினோராவது ஆளாகக் களம் இறங்குபவராக இருந்தால் எந்தவித உத்திகளுக்கும் தேவையே இருக்காது. சரி தானே! ஆக, சிக்கலான செயல்களை எளிதாக முடிப்பதற்குத் தான் உத்திகள் தேவைப்படுகின்றன.

டெஸ்ட் கேஸ் எழுதுவது என்பது சிக்கலான ஒன்றா?

டெஸ்ட் கேஸ் எழுதுவது சிக்கலான ஒன்று இல்லை தான்! ஆனால் பயனர் எந்தெந்த விதமான உள்ளீடுகளை எல்லாம் கொடுப்பார் என்று சிந்தித்துச் சிந்தித்து டெஸ்ட் கேஸ்களை எழுதுவது என்பது அதிக நேரத்தை எடுக்கும் செயல்! எனவே விரயமாகும் நேரத்தை மிச்சப்படுத்திக் குறைந்த நேரத்தில் நேர்த்தியாக டெஸ்ட் கேஸ்களை உருவாக்கி, அதிக நேரத்தை உண்மையான சோதித்தலில் செலவழிப்பது தான் நிறைய பிழைகளைக் கண்டுபிடிக்க உதவும்! எவ்வளவுக்கு எவ்வளவு பிழைகளைக் கண்டுபிடிக்கிறோமோ அந்த அளவு மென்பொருளின் தரம் உயரும்! அதற்காகக் தான் டெஸ்ட் கேஸ்களுக்கு உத்திகள் வகுத்துச் செயல்படுகிறோம். அவற்றில் சிலவற்றை இப்போது பார்ப்போம்.

1. எல்லைகளைச் சோதிப்போம்: :

மட்டைப் பந்துப் போட்டி நடந்து கொண்டிருக்கிறது. மட்டையாளர் பந்தைத் தூக்கி அடிக்கிறார். அடித்த பந்து, பார்வையாளர்களுக்கு நடுவில் நேரே சென்று விழுகிறது. இப்போது நடுவர் எளிதாக 'ஆறு' எனக் கைகளைத் தூக்கிக் காட்டிவிட முடியும். அடுத்த பந்து - மட்டையாளர் மீண்டும் முழுவீச்சில் அடிக்கிறார் - பந்து, இப்போது எல்லைக் கோட்டை ஒட்டிப் போய் விழுகிறது. இப்போது நடுவர் முன்பு கொடுத்தது போல, எளிதாக நான்கு அல்லது ஆறு எனக் கொடுத்து விட முடியாது. மூன்றாவது நடுவரிடம் கலந்து பேசி, பந்து

- எல்லைக் கோட்டிற்கு அப்பால் விழுந்ததா
- எல்லைக் கோட்டின் மேல் விழுந்ததா
- எல்லைக் கோட்டிற்கு உள்பக்கம் விழுந்ததா

என்று தெரிந்து கொண்டு முடிவை அறிவிப்பார் அல்லவா? எல்லைகளை நெருங்கும் போது இவை போன்ற சிக்கல்கள் வருவது இயல்பு!

எல்லைச் சிக்கலும் பேருந்தில் அரைக்கட்டணமும் :

நாம் பேருந்தில் பயணித்துக் கொண்டிருக்கிறோம் என்று வைத்துக் கொள்ளுங்கள். பேருந்தில் பன்னிரண்டு வயதுக்கு உட்பட்ட அல்லது 135 செ.மீ உயரத்திற்குக் கீழ் உள்ள குழந்தைகளுக்கு அரைக்கட்டணம் வாங்குகிறார்கள். இங்கு, நடத்துநருக்கு,

- 4 வயதுக் குழந்தைக்கு அரைக்கட்டணம் வாங்குவதில் சிக்கல் இருக்காது
- 15 வயது சிறுவனுக்கு முழுக்கட்டணம் வாங்குவதிலும் எந்தச் சிக்கலும் இருக்காது.

எங்குச் சிக்கல் வரும்?

அரைக்கட்டணத்திற்கு வந்து நிற்கும் குழந்தையோ சிறுவனோ

- பன்னிரண்டு வயதோ அதை ஒட்டியோ உள்ள வயதினர் போல் தோற்றமளிக்கும் போது
- 135 செ.மீ. உயரத்தை ஒட்டி அவர்கள் உயரம் அமையும் போது

இந்த இரண்டு இடங்களில் தான் நடத்துநர் குழம்பிப் போவார். இதே கதை தான் மென்பொருளிலும்!

எடுத்துக்காட்டாக, ஐந்து இலட்சம் வரை சம்பளம் இருந்தால் வருமான வரி கிடையாது என்று வாடிக்கையாளர் தேவை ஆவணத்தில் குறிப்பிட்டிருப்பார்கள். நம்முடைய உருவாக்குநர், ஐந்து இலட்சம் வருமானம் ஈட்டும் ஒருவருக்கு வருமான வரி உண்டா? இல்லையா? என்பதில் குழம்பிப் போய் இருப்பார்.

எனவே, இந்த எல்லைகளை முறைப்படிச் சோதிப்பது தான் எல்லைச் சோதனை முறையாகும்.

மேலும் சில எ.கா. கள்:

- பத்து இலக்க அலைபேசி எண்ணைக் கேட்கும் உரைப் பெட்டிகளில் முறையே,
    - ஒன்பது இலக்க எண்ணைக் கொடுத்துச் சோதிப்பது
    - பத்து இலக்க எண்ணைக் கொடுத்துச் சோதிப்பது
    - பதினோர் இலக்க எண்ணைக் கொடுத்துச் சோதிப்பது

ஆகிய மூன்றிற்கும் டெஸ்ட் கேஸ்களை எழுதி வைத்துக் கொள்வது! முறைப்படி இந்தச் சோதனையை நாம் செய்வதாக இருந்தால், ஓரிலக்க எண்ணில் தொடங்கி, ஈரிலக்க எண், மூவிலக்க எண் எனத் தொடர்ந்து, பதினைந்து, இருபது இலக்க எண்கள் வரை சோதிப்பதற்கு டெஸ்ட் கேஸ்கள் எழுத வேண்டும். அதற்கு எளிய மாற்றாக,

பத்து இலக்க எண்ணைச் சோதிக்க,

- பத்து இலக்க எண்ணைக் கொண்டு ஒரு சோதனை - முறையான உள்ளீடு
- ஒன்பது இலக்க எண்ணைக் கொண்டு ஒரு சோதனை - முறையற்ற உள்ளீடு (முறையான உள்ளீட்டிற்குக் கீழ்)
- பதினோர் இலக்க எண்ணைக் கொண்டு ஒரு சோதனை - முறையற்ற உள்ளீடு (முறையான உள்ளீட்டிற்கு மேல்)

ஆகிய மூன்று டெஸ்ட் கேஸ்களை மட்டுமே எழுதி வைத்துக் கொள்வது தான் எல்லைச் சோதனை உத்தியாகும். இதை ஆங்கிலத்தில் 'பவுண்டரி வேல்யூ அனாலிசிஸ்' ('Boundary Value Analysis') என்று சொல்வார்கள்.

2. சரிவிகிதப் பிரிப்பு முறை :

நாம் ஒரு தெருவில் நடந்து போய்க் கொண்டிருக்கிறோம் என்று வைத்துக் கொள்ளுங்கள். தெருவின் இரு பக்கங்களிலும் இருபது இருபது வீடுகள் இருக்கின்றன. ஒரு பக்கத்தில் வீட்டு எண்கள் ஒன்றில் தொடங்கி இருபதில் முடிகின்றன. மறு பக்கத்தில் வீட்டு எண்கள் இருபத்து ஒன்றில் தொடங்கி, நாற்பதில் முடிகின்றன. உங்களுக்குக் கொடுக்கப்பட்டிருக்கும் வேலை, முப்பதாவது எண்ணுடைய வீட்டைக் கண்டுபிடிக்க வேண்டும். என்ன செய்வீர்கள்?

இரு எளிய தீர்வுகள் :

அ. தெருவில் ஒரு பக்கத்தைப் பார்த்துக் கொண்டே முதல் இருபது வீடுகளைப் பார்த்து மறு முனையை அடைந்து, மீண்டும் அங்கிருந்து மறுபக்கத்தில் இருக்கும் வீடுகளில் முப்பதாம் எண்ணுடைய வீட்டைக் கண்டுபிடிக்கலாம்.
ஆ. தெருவில் வீட்டு எண்கள் எப்படிக் கொடுக்கப்பட்டிருக்கின்றன என்பதை ஊகித்து, தெருவின் மறுபக்கத்தில் இருக்கும் முப்பதாம் எண்ணுடைய வீட்டைக் கண்டுபிடிக்கலாம்.
இந்த இரண்டு தீர்வுகளில் எந்தத் தீர்வு எளிதானது? இதென்ன கேள்வி - இரண்டாவது தீர்வு தானே! என்கிறீர்களா? சரி தான்!
இதே போல் ஒரு தீர்வை டெஸ்ட் கேஸ் உருவாக்கத்தில் பின்பற்றுவதற்காகத் தான் சரிவிகிதப் பிரிப்பு உத்தியைப் பார்க்கிறோம். மேலே, நாம் பார்த்த எடுத்துக்காட்டைப் போல, பயனர் கொடுக்கவிருக்கும் உள்ளீட்டைச் சரிவிகிதத்தில் பிரித்துக் கொள்வார்கள்.
காப்பீட்டுக் கழகம் ஒன்றிற்குரிய மென்பொருளை நம் நிறுவனம் உருவாக்குவதாக வைத்துக் கொள்ளுங்கள். காப்பீட்டுக் கழகம், வாடிக்கையாளரின் வயது 18 வயதில் இருந்து 60 வயது வரை இருந்தால் சிறப்புச் சலுகை ஒன்றைக் கொடுக்கிறது என்று வாடிக்கையாளர் தேவை ஆவணம் சொல்கிறது. இந்தச் சோதனைக்கு எத்தனை டெஸ்ட் கேஸ்கள் எழுதுவது?

- ஒரு வயதில் இருந்து பதினேழு வயது வரை பதினேழு டெஸ்ட் கேஸ்கள்
- பதினெட்டு வயதில் இருந்து அறுபது வயது வரை நாற்பதிற்கும் அதிகமான டெஸ்ட் கேஸ்கள்
- அறுபத்தோரு வயதில் இருந்து இன்னும் எண்ணற்ற டெஸ்ட் கேஸ்கள்

என்று எழுதினால் டெஸ்ட் கேஸ்களை எழுதி முடிப்பதற்குள்ளாகவே நம்முடைய வயதும் மாறி, வாடிக்கையாளரின் வயதும் மாறிவிடும்! அவ்வளவு நேரம் பிடிக்கும். இது போன்ற சூழலில்,
18 – 60 வயது என்பது தான் தகுதியான உள்ளீடு. எனவே

- 18 வயதுக்கு முன் ஏதேனும் ஒரு வயதுக்கு டெஸ்ட் கேஸ்
- 18 – 60 வயது வரம்புக்குள் ஏதேனும் ஒரு வயதுக்கு டெஸ்ட் கேஸ்

- 60 வயதுக்கு மேல் ஏதேனும் ஒரு வயதுக்கு டெஸ்ட் கேஸ்

என்று மூன்றே மூன்று டெஸ்ட் கேஸ்கள் எழுதிச் சோதனையைச் செய்வது தான் கால விரயத்தையும் தவிர்க்கும்; அதே வேளையில் பயனர் கொடுக்கப் போகும் எல்லாவித உள்ளீட்டையும் சோதிக்கவும் உதவும். அதாவது,

- தகுதியான எல்லையில் உள்ளீடு கொண்டு ஒரு டெஸ்ட் கேஸ்
- தகுதி எல்லைக்கு மேல் ஒரு டெஸ்ட் கேஸ்
- தகுதி எல்லைக்குக் கீழ் ஒரு டெஸ்ட் கேஸ்

இந்த உத்தியைத் தான் சரிவிகிதப் பிரிப்பு உத்தி என்கிறோம். ஆங்கிலத்தில் இந்த உத்திக்கு 'ஈக்குவலன்ஸ் கிளாஸ் பார்ட்டிஷனிங்' ('Equivalence Class Partitioning') என்று பெயர்.

இதுவரை நாம் பார்த்ததில் இருந்து, டெஸ்ட் கேஸ்கள் எழுதும் வேலையை எளிமையாக்க உத்திகள் வெகுவாக உதவுகின்றன என்பது தெளிவாகி விட்டது. இப்போது சொல்லுங்கள் - இந்த இரண்டு உத்திகள் போதுமா? இல்லை டெஸ்ட் கேஸ்கள் உருவாக்கத்தை எளிமையாக்க இன்னும் உத்திகள் வேண்டுமா? வேண்டும் தானே! மேலும் சில உத்திகளை வரும் பதிவுகளில் பார்ப்போம்!

டெஸ்ட் கேஸ் உத்திகள் - 2

அலைபேசி, கணினி, தொலைக்காட்சி, குளிரூட்டி என ஏராளமான மின்னணுக் கருவிகள் விற்கும் நிறுவனம் ஒன்றில் வாடிக்கையாளர்களுக்கு இரசீது கொடுக்கும் மென்பொருள் ஒன்றை நம்முடைய உருவாக்குநர்கள் உருவாக்கிக் கொடுப்பதாக வைத்துக்கொள்ளுங்கள். அவர்கள் உருவாக்கும் நேரத்தில் சோதனையாளர்கள் டெஸ்ட் கேஸ்கள் எழுதத் தொடங்கியிருப்பார்கள். இங்கு நம்முடைய மென்பொருள் இரசீது கொடுக்கும் மென்பொருள் என்பதால், பல்வேறு வகைகளில் இரசீதுகளைச் சோதிக்க டெஸ்ட் கேஸ்கள் உருவாக்கப்பட வேண்டும்.

அதாவது,

- 10000 ரூபாய்க்கு மேல் பொருள் வாங்கினால் தள்ளுபடி
- கடன் அட்டை (கிரெடிட் கார்டு) கொண்டு வாங்கினால் சிறப்புத் தள்ளுபடி
- குறிப்பிட்ட நாட்களில் வாங்கினால் சிறிய தள்ளுபடி
- இரண்டு அல்லது அதற்கு மேல் பொருட்கள் வாங்கினால் இலவசப் பொருள்
- குறிப்பிட்ட நிறுவனங்களின் பொருட்களை வாங்கினால் குறைந்த விலை

என்று பல்வேறு தள்ளுபடிகளையும், இலவசங்களையும் இரசீதில் சோதிக்கும் வகையில் டெஸ்ட் கேஸ்கள் எழுதப்பட வேண்டும். இப்படிப்பட்ட டெஸ்ட் கேஸ்கள் எழுதுவதில் மிகுந்த அக்கறை வேண்டும். ஏனென்றால், சோதனையாளர் கவனிக்காமல் விடும் ஒவ்வொரு டெஸ்ட் கேசும் இரசீதில் தவறுகளைக் கொண்டு வந்து விடும்; அது நிறுவனத்தின் வணிகத்தை நேரடியாகப் பாதித்து விடும் அல்லவா? எனவே இது போன்ற இடத்தில் டெஸ்ட் கேஸ்களைக் கூடுதல் கவனத்துடன் எழுத வேண்டும். ஏனென்றால், மேலே பார்த்த தள்ளுபடிகள் பல சமயங்களில் கலவையாகக் கொடுக்கப்படும். அதாவது,

- 10000 ரூபாய்க்கு மேலும் வாங்கிக் குறிப்பிட்ட வங்கிக் கடன் அட்டையைப் பயன்படுத்தினால் சிறப்புச் சலுகை
- குறிப்பிட்ட நாட்களில் குறிப்பிட்ட சில நிறுவனங்களின் பொருட்களை வாங்கினால் மேல் அதிகத் தள்ளுபடி விலை

என்பன போன்று! இவற்றை எல்லாம் சேர்த்து டெஸ்ட் கேஸ்களை எழுத வேண்டும். எழுதப் படும் டெஸ்ட் கேஸ்களில் மேலே பார்த்தது போல் இருக்கும் கலவைகள் எவையும் தவறிவிடக் கூடாது. நம்முடைய கையில் இப்போது இரண்டு பெரிய வேலைகள் இருக்கின்றன.

1) நிறைய கலவைகளைச் சோதிப்பதற்கு டெஸ்ட் கேஸ்கள் எழுத வேண்டும்.

2) அக்கலவைகளுக்குரிய தள்ளுபடி சரியான வகையில் 'எதிர்பார்க்கும் வெளியீடாக' அமைய வேண்டும்.

இவை போன்ற நேரங்களில் தான் உள்ளீடுகளையும் அவற்றிற்குரிய முடிவுகளையும் ஓர் அட்டவணை போல் உருவாக்கி, அந்த அட்டவணையின் அடிப்படையில் டெஸ்ட் கேஸ்களை எழுதுவது நேரத்தையும் மிச்சமாக்கும்; எல்லாவகை உள்ளீட்டையும் ஒன்று விடாமல் டெஸ்ட் கேசாக மாற்றிவிட உதவும்.

12	1	2	3	4	5	6	7
10000 ரூ. க்கு மேல்	ஆம்	ஆம்	ஆம்	ஆம்	ஆம்	இல்லை	இல்லை
கடன் அட்டை	ஆம்	ஆம்	ஆம்	ஆம்	இல்லை	ஆம்	ஆம்
குறிப்பிட்ட பொருள்	ஆம்	ஆம்	ஆம்	இல்லை	ஆம்	ஆம்	இல்லை

குறிப்பிட்ட வங்கி	ஆம்	ஆம்	இல்லை	ஆம்	ஆம்	ஆம்	ஆம்
2 பொருளுக்கு மேல்	ஆம்	ஆம்	ஆம்	ஆம்	ஆம்	ஆம்	ஆம்
குறிப்பிட்ட நாள்	ஆம்	இல்லை	ஆம்	ஆம்	ஆம்	ஆம்	இல்லை
எதிர்பார்க்கும் முடிவுகள்							
5 % தள்ளுபடி						உண்டு	உண்டு
கடிகாரம் இலவசம்	உண்டு		உண்டு		உண்டு		
10% தள்ளுபடி		உண்டு	உண்டு		உண்டு		
20% தள்ளுபடி	உண்டு			உண்டு			

மேல் இருப்பது போல, வாடிக்கையாளர் தேவை ஆவணத்தை அடிப்படையாகக் கொண்டு - விதிகளையும் எதிர்பார்க்கும் முடிவுகளையும் அட்டவணைப்படுத்த வேண்டும். இப்படி அட்டவணைப்படுத்துவதென்பது, முன்னரே சொன்னது போல, எல்லாவகை உள்ளீடுகளையும் சோதிக்க மிகுந்த உதவியாக இருக்கும். இந்த முறைக்கு முடிவுகளை அட்டவணைப்படுத்தும் முறை ('Decision Table') என்று பெயர்.

நிலை மாற்ற முறை சோதனை

சில சமயங்களில் பயனர் கொடுக்கும் உள்ளீடுகள் ஒன்றாக இருக்கும். ஆனால் கிடைக்கும் விடைகள் ஒவ்வொரு முறை ஒவ்வொன்றாக இருக்கும். ஏடிஎம் மையத்திற்குச் சென்று உங்களுடைய கடவுச்சொல்லை உள்ளிடுவதை எடுத்துக்கொள்ளுங்கள். முதல் இரண்டு முறை தவறான கடவுச்சொல்லை உள்ளிடும் போது ஏடிஎம் கருவி, 'தவறான கடவுச்சொல்'

என்று பதில் சொல்லும். மூன்றாவது முறையும் தவறான கடவுச்சொல்லை உள்ளிட்டால், 'கடவுச்சொல் தவறு - 24 மணி நேரம் காத்திருக்கவும் அல்லது அருகில் உள்ள வங்கிக்கிளையை அணுகவும்' என்று செய்தி தரும்.

இந்த எடுத்துக்காட்டில், மூன்று முறையும் பயனர் ஒரே வேலையைத் (தவறான கடவுச்சொல் கொடுத்ததைத்) தான் செய்தார். ஆனால் முதல் இரண்டு முறை கிடைத்த பதில், மூன்றாவது முறை அமையவில்லை. , பயனர் பயன்படுத்தியது போலவே ஒரு சோதனையாளராக நாமும் மூன்று முறை இது போன்ற நிகழ்வில் சோதிக்க வேண்டும். இங்கு, தொடக்க நிலையில் ஏடிஎம் கருவி தன்னியல்பில் இருக்கிறது. முதல் இரண்டு உள்ளீடுகளால் அதன் நிலையில் மாற்றம் ஏதுமில்லை. ஆனால் மூன்றாவது உள்ளீட்டால் ஏடிஎம் கருவி தன் நிலையை மாற்றி, இனி அட்டையை உள்ளிட்டாலும் செயல்பட மறுக்கிறது அல்லவா? இதைத் தான் நிலை மாற்ற முறை சோதனை (State Transition) என்று சொல்வார்கள். இதையும் டெஸ்ட் கேஸ்கள் எழுதும் போது கருத்தில் கொள்ள வேண்டும்.

இதுவரை நாம் கண்ட உத்திகள் அனைத்தும் கருப்புப் பெட்டிச் சோதனை முறையில் டெஸ்ட் கேஸ்களை எழுதப் பயன்படும் உத்திகள் ஆகும். அதென்ன கருப்புப் பெட்டிச் சோதனை? விமானத்தில் தான் கருப்புப் பெட்டி என்று கேட்டிருக்கிறோம். இதென்ன மென்பொருளில் கருப்புப்பெட்டிச் சோதனை என்கிறீர்களா? கருப்புப் பெட்டி மட்டுமில்லை, வெள்ளைப் பெட்டிச் சோதனையும் இருக்கிறது. வரும் பதிவுகளில் அவற்றைப் பற்றிக் கலந்துரையாடுவோம்.

கருப்புப் பெட்டியும் வெள்ளைப் பெட்டியும்

வானூர்தியில் தான் கருப்புப் பெட்டி என்று கேள்விப்பட்டிருக்கிறேன். இதென்ன சாப்ட்வேர் டெஸ்டிங்கிலும் கருப்புப் பெட்டியா? என்று வியக்கிறீர்களா? வியக்க வேண்டாம். எளிமையானது தான்! பார்த்து விடலாமா?

வானூர்தியில் தான் கருப்புப் பெட்டி என்று கேள்விப்பட்டிருக்கிறேன். இதென்ன சாப்ட்வேர் டெஸ்டிங்கிலும் கருப்புப் பெட்டியா? என்று வியக்கிறீர்களா? வியக்க வேண்டாம். எளிமையானது தான்! பார்த்து விடலாமா?

கருப்புப் பெட்டிச் சோதனை:

வீட்டில் இருக்கும் மோடத்திற்கு (Modem) இணைய இணைப்புக் கொடுக்கிறீர்கள். ஆனால் அந்த மோடம் எப்படி உள்ளீட்டை வாங்குகிறது? எப்படி உங்களுக்கு இணைய வசதி கிடைக்கிறது என்று யோசித்திருக்கிறீர்களா? எப்படிக் கிடைத்தால் என்ன – இணையம் கிடைத்தால் போதும் அல்லவா?

வண்டிக்குப் பெட்ரோல் நிரப்புகிறோம். பெட்ரோல் இஞ்சினுக்குப் போய் இஞ்சினை இயக்குகிறது. எப்படி இது நடக்கிறது என்று என்றைக்காவது யோசித்திருக்கிறோமா? இல்லை அல்லவா? உங்கள் மோடம், இஞ்சின் ஆகியவற்றைக் கருப்புப் பெட்டி என்று

நினைத்துக் கொள்ளுங்கள். அதாவது, உள்ளீடு என்ன என்று தெரியும் - வெளியீடு என்ன என்று தெரியும் - உள்ளே என்ன நடக்கிறது என்பது மட்டும் தெரியாது. இது தான் கருப்புப் பெட்டிச் சோதனை முறை!

அதாவது, டெவலப்பர்கள் உருவாக்கித் தரும் மென்பொருளுக்கு என்ன உள்ளீடு கொடுக்க வேண்டும், அந்த உள்ளீட்டுக்கு என்ன வெளியீடு கிடைக்க வேண்டும் என்பதை மட்டும் பார்த்தால் போதும்! அந்த மென்பொருளை எப்படி உருவாக்கியிருக்கிறார்கள், என்ன நிரல் மொழி பயன்படுத்தியிருக்கிறார்கள் என்பதைப் பற்றியெல்லாம் கவலைப்படத் தேவையில்லை! இது தான் கருப்புப் பெட்டிச் சோதனை முறை! நாம் முந்தைய இரண்டு பதிவுகளில் பார்த்த உத்திகள் அனைத்தும் இந்தச் சோதனை முறைக்கு உரிய உத்திகள் தாம்! கருப்புப் பெட்டிச் சோதனை சரி! வெள்ளைப் பெட்டி என்றால் என்ன என்கிறீர்களா? கருப்புக்கு நேர் எதிர் தானே வெள்ளை! ஆமாம்! நாம் சோதிக்கப் போகும் மென்பொருளை எப்படி உருவாக்கியிருக்கிறார்கள் - நிரல் மொழி என்ன பயன்படுத்தியிருக்கிறார்கள்? எப்படி நிரல் எழுதியிருக்கிறார்கள் என்பன போன்ற அனைத்தையும் தெரிந்து அவற்றைச் சோதிப்பதையும் சேர்ந்து செய்வது தான் வெள்ளைப் பெட்டி முறை! இந்த வெள்ளைப் பெட்டி முறைக்குத் திறந்த பெட்டி முறை, கண்ணாடிப் பெட்டி முறை என்று வேறு சில பெயர்களும் உள்ளன.

கருப்புப் பெட்டிச் சோதனை முறையில் சில உத்திகள் சொன்னீர்களே! அப்படியானால் அப்படிப்பட்ட உத்திகள் வெள்ளைப் பெட்டி முறையிலும் இருக்க வேண்டுமே! என்று கேட்கிறீர்களா? கவலையே படாதீர்கள். நிரலை வரிவரியாகச் சோதிப்பது, நிரலின் போக்கு அடிப்படையில் சோதிப்பது, என்று இங்கும் சில உத்திகள் இருக்கின்றன. இந்த உத்திகளைச் செயல்படுத்த வேண்டும் என்றால், டெஸ்டர்களுக்கு நிரல் மொழி (புரோகிராமிங் லேங்குவேஜ்) பற்றிய புரிதல் இருக்க வேண்டுமே என்று நினைக்கிறீர்களா? உங்கள் நினைப்புச் சரி தான்! கட்டாயம் வெள்ளைப் பெட்டி முறை டெஸ்டர்களுக்கு மொழி பற்றிய புரிதல் இருக்க வேண்டும். வெறுமனே திட்டப் பணி பற்றிய புரிதல் மட்டும் போதாது.

வெள்ளைப் பெட்டிச் சோதனை முறையில் இருக்கும் உத்திகள் என்னென்ன? அடுத்த பதிவில் அதைப் பார்ப்போமா!

வெள்ளைப் பெட்டி உத்திகள்

வெள்ளைப் பெட்டி என்று சாப்ட்வேர் டெஸ்டிங்கில் எதைச் சொல்கிறார்கள்? கருப்புப் பெட்டி என்றால் என்று பார்த்தோம் அல்லவா? அதற்கு நேர் எதிரானது தான் வெள்ளைப் பெட்டி! வெளிப்படையான (transparent) பெட்டியைத் தான் வெள்ளைப் பெட்டி என்று சொல்கிறார்கள். வெளிப்படை என்றால் என்ன? கணினியில் இரண்டு எண்களைக் கூட்டுவதற்கு நிரல்(program) எழுதுகிறீர்கள் என்று வைத்துக் கொள்ளுங்கள். நிரலின் முடிவில் இரண்டு எண்களைக் கொடுக்கிறீர்கள். வரும் வெளியீடு(output) சரியா என்று பார்க்கிறீர்கள். இது தான் கருப்புப் பெட்டி முறை! அதாவது உள்ளீட்டுக்குத் தகுந்த வெளியீடு கிடைக்கிறதா என்று மட்டும் பார்ப்பது! வெள்ளைப் பெட்டி முறையில் உள்ளீடு,

வெளியீடு ஆகியவற்றோடு நீங்கள் எழுதியிருக்கும் நிரலையும் சேர்த்துச் சரிபார்ப்பது வெள்ளைப் பெட்டி முறை!
அதாவது,
* உள்ளீடு
* வெளியீடு ஆகியன மட்டுமல்லாது
* நிரல் எப்படி எழுதியிருக்கிறார்கள்
* கட்டுப்பாட்டு வாக்கியங்களை எப்படி அமைத்திருக்கிறார்கள்
* திருப்பு வாக்கியங்களை எப்படி அமைத்திருக்கிறார்கள்
என்பனவற்றையும் சேர்த்துச் சோதிப்பது தான் வெள்ளைப் பெட்டி முறையாகும்.
ஆகா! திடீரென கட்டுப்பாட்டு வாக்கியங்கள், திருப்பு வாக்கியங்கள் என்றெல்லாம் பேசுகிறீர்களே! என்னவென்றே புரியவில்லையே! என்கிறீர்களா? கவலையை விடுங்கள்! தெலுங்கில் ஒரு பழமொழி உண்டு - ‘பேரு பெத்த பேரு - தாக நீரு லேது!’ என்று! அதாவது - பெரிய ஆளாக இருப்பார் - ஆனால் குடிக்கத் தண்ணீர் கூடத் தரமாட்டார்! தொழில்நுட்ப வார்த்தைகளும் அப்படிப்பட்டவை தான்! பார்க்கப் பெரிதாக பயமுறுத்துவது போலத் தெரியும்! உள்ளே நுழைந்து கொஞ்சம் ஆராய்ந்தால் - ‘அட! இது தானா!’ என்று நினைப்போம். எனவே, கட்டுப்பாட்டு வாக்கியங்கள், திருப்பு வாக்கியங்கள் ஆகியனவற்றைப் பற்றி விரிவாகப் பின்னர் பார்ப்போம். இப்போதைக்கு அவை எல்லாம் எளிமையானவை தான் என்று மட்டும் மனத்தில் ஏற்றிக் கொள்ளுங்கள்.

நிரல் பற்றித் தெரிய வேண்டுமா?

ஆனால் ஒன்று! வெள்ளைப் பெட்டி முறை என்பதே உள்ளீடு, வெளியீடு ஆகியவற்றுடன் நிரலையும் ஆராய்வது என்பதால், இந்த முறையில் சோதிப்பதற்கு, சோதனையாளருக்கு (டெஸ்டருக்கு) நிரல் பற்றிய அறிவு கட்டாயம் தேவை!
நிரலை எடுத்துச் சோதிப்பதன் மூலம் மென்பொருள் சரியாக இயங்குகிறதா என்று பார்ப்பதுடன் மெமரி லீக் (நினைவக ஓட்டை) இருக்கிறதா என்று கூடப் பார்க்கலாம்.
இந்தச் சோதனையைச் செய்வதற்கு வழக்கம் போல் கட்டற்ற மென்பொருட்கள் ஸ்பிலின்ட், வால்கிரைண்டு என்பன போன்று நிறைய இருக்கின்றன.
எல்லாம் சரி! இந்தப் பதிவின் தலைப்பு என்னவோ வெள்ளைப் பெட்டி உத்திகள் என்று சொல்கிறது - ஆனால் நீங்கள் இன்னும் உத்திகளைப் பற்றிப் பேசவே இல்லையே! என்று தோன்றுகிறதா?
வெள்ளைப் பெட்டி உத்திகள்:
1) வரிவரிச் சோதனை முறை (Statement Coverage)
2) கிளைவரிச் சோதனை முறை (Branch Coverage)
3) வழிச் சோதனை முறை (Path Coverage)
ஆகிய உத்திகளைப் பயன்படுத்தி வெள்ளைப் பெட்டிச் சோதனையில் ஈடுபடலாம். இவை தவிர்த்து, மாற்றச் சோதனை முறையும் (Mutation Testing) உண்டு. அவற்றைப் பற்றித் தான் அடுத்த பதிவு! விரிவாகப் பேசுவோம்.

வெள்ளைப் பெட்டி உத்திகள் -2

போன பதிவில் வெள்ளைப் பெட்டி என்றால் என்ன என்பது பற்றியும் அதன் உத்திகள் என்னென்ன என்பதையும் பார்த்தோம். இப்போது நாம் பார்க்கவிருப்பது அந்த உத்திகளைப் பற்றித் தெரிந்து கொள்வது தான்! போன பதிவில் என்னென்ன உத்திகளைப் பற்றிப் பேசினோம் என்று நினைவில் இருக்கிறதா? ஆம்!

1) வரிவரிச் சோதனை முறை (Statement Coverage)
2) கிளைவரிச் சோதனை முறை (Branch Coverage)
3) வழிச் சோதனை முறை (Path Coverage)
4) மாற்றச் சோதனை முறையும் (Mutation Testing) ஆகிய நான்கும் தான்! ஒவ்வொன்றாகப் பார்ப்போமா?

1) வரிவரிச் சோதனை முறை (Statement Coverage)

பேரே பொருளைச் சொல்லிவிட்டதல்லவா? வரி விடாமல் ஒவ்வொரு வரியையும் எடுத்துச் சோதிப்பது தான் வரிவரிச் சோதனை முறையாகும். அதாவது, நிரலின் ஒவ்வொரு வரியும் சரியாக இயங்குகிறதா என்று பார்ப்பது! எளிமையான ஒரு நிரலைப் பார்ப்போமா?

```
1 நிரலின் தொடக்கம்
2 முதல் எண்ணை வாங்குக.
3 2 ஆவது எண்ணை வாங்குக.
4 முதல் எண், 2 ஆவது எண்ணை விடப் பெரிதாக இருந்தால்,
அகர முதல எழுத்தெல்லாம் என்று சொல்க
5 நிரலின் முடிவு
```

இங்கு மொத்தம் 5 வரிகள் உள்ளன. இந்த ஐந்து வரிகளையும் சோதிக்க வேண்டும் என்றால், முதல் எண், 2 ஆவது எண்ணை விடக் கட்டாயம் பெரிய எண்ணாகத் தான் இருக்க வேண்டும். (அப்போது தானே ‘அகர முதல எழுத்தெல்லாம்’ என்பது வரும்!).

இன்னொரு நிரலைப் பார்ப்போம்!

```
1 நிரலின் தொடக்கம்
2 முதல் எண்ணை வாங்குக
3 2 ஆவது எண்ணை வாங்குக
4 3 ஆவது எண் = முதல் எண் + (2 * 2 ஆவது எண்)
5 3 ஆவது எண் 100 ஐ விடப் பெரிதாக இருந்தால்
மின்மினி எனச் சொல்க
6 நிரலின் முடிவு
```

இங்கு வரிவரிச் சோதனை செய்ய என்னவெல்லாம் செய்யலாம்? முதல் எண்ணையும் 2 ஆவது எண்ணையும் கொடுத்துப் பார்ப்போமா?

முயற்சி #1:

முதல் எண் = 20, 2 ஆவது எண் = 30 எனக் கொள்வோம்.

இப்போது 3 ஆவது எண் = 20 + (2* 30) – அதாவது 80. இந்த உள்ளீடுகளுக்கு 5 ஆவது வரி செயல்படாது தானே!

இந்த முயற்சியில் எல்லா வரிகளும் இயக்கப்படவில்லை.

முயற்சி #2:

முதல் எண் = 30, 2 ஆவது எண் = 40 எனக் கொள்வோம்.

இப்போது 3 ஆவது எண் = 30 + (2* 40) – அதாவது 110. இந்த உள்ளீடுகளுக்கு 5 ஆவது வரி செயல்படும்.

இந்த முயற்சியில் எல்லா வரிகளும் இயக்கப்பட்டிருக்கின்றன. எனவே, முயற்சி #2 ஐ வரிவரிச் சோதனைக்குரிய சரியான முயற்சியாகும்.

2) கிளைவரிச் சோதனை முறை **(Branch Coverage)**

கிளைவரிச் சோதனையைப் புரிந்து கொள்ள, ஒரு நிரலில் கிளை வரி (Branch) என்றால் என்ன என்பதை முதலில் புரிந்து கொள்வது அவசியம். சென்னையில் இருந்து மதுரை போவது தான் நம்முடைய நிரல் என்று வைத்துக் கொள்ளுங்கள். சென்னையில் இருந்து திருச்சிராப்பள்ளி வழியாக மதுரைக்குப் போகலாம். சென்னையில் இருந்து சிதம்பரம், சீர்காழி, மயிலாடுதுறை, கும்பகோணம், தஞ்சாவூர் என்று ஊரையெல்லாம் சுற்றித் திருச்செந்தூர் விரைவான் போவது போல், இன்னொரு வழியில் மதுரைக்குப் போகலாம். ஆக, நம்முடைய சென்னை – மதுரை நிரலுக்கு இப்போது இரண்டு கிளைகள். ஒன்று நேர் வழி, இன்னொன்று சுற்று வழி! இதைத் தான் கிளை வரி(Branch) என்கிறோம்.

ஒருவர் சென்னையில் இருந்து மதுரைக்குப் போவதைக் கிளைவரிச் சோதனை செய்ய வேண்டும் என்றால் இப்போது இந்த இரண்டு கிளைகளையும் சோதிக்க வேண்டும். இந்தக் கிளைவரிகளின் எண்ணிக்கை கூடக் கூட, சோதனையின் கடினத் தன்மை கூடும் – அதாவது சோதிக்க நீண்ட நேரம் ஆகும். சென்னையில் இருந்து மும்பைக்குப் போவதை,

1) பேருந்து வழிப் பயணம்

2) கடல்வழிப் பயணம்

3) வான்வழிப் பயணம்

என மூன்று கிளைகளில் சோதிக்கலாம் எனில் இங்கு, கிளைவரிச் சோதனையின் எண்ணிக்கை 3 – சரி தானே!

இப்போது, கிளை வரிச் சோதனை புரிந்திருக்கும் என நினைக்கிறேன். வரிவரிச் சோதனை, கிளை வரிச் சோதனை ஆகிய இரண்டுமே இப்போது உங்களுக்குத் தெரியும் அல்லவா? இரண்டையும் சேர்த்துப் பார்த்தால் ஏதாவது தோன்றுகிறதா? “ஆமாம்! ஆமாம்! கிளைவரிச் சோதனையை நூறு விழுக்காடு முடித்தால் வரிவரிச் சோதனையையும் நூறு விழுக்காடு முடித்தது போல் தான்! சரிதானே! கணியம் வாசகர்களா? கொக்கா?” என்கிறீர்களா? க க க போங்கள்! இவ்வளவு கூர்மையான ஆளாக நீங்கள் இருந்தால் – வழிச் சோதனை என்றால் என்ன என்றும் பார்த்து விடுவோமா? அடுத்த பதிவில் பார்ப்போம்!

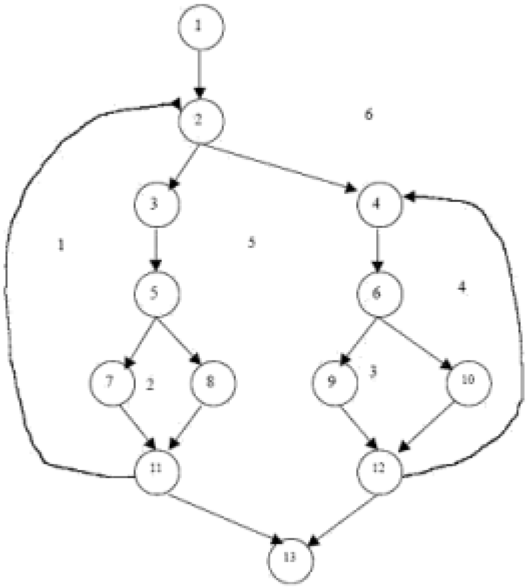
வெள்ளைப் பெட்டி உத்திகள் – 3

இந்தப் பதிவில் நாம் பார்க்கவிருப்பது வழிச் சோதனை முறை தான்!

3) வழிச் சோதனை முறை (Path Coverage)

ஒரு நிரலின் எல்லா வழிகளையும் சோதித்துப் பார்ப்பது தான் வழிச் சோதனை முறை ஆகும்.

நாம் இது வரை பார்த்த சோதனை முறைகளை எல்லாம் வைத்து சுழல் முறை கடினத்தன்மை (‘Cyclomatic Complexity’)யைக் கண்டுபிடிக்கலாம். அதென்ன ‘சுழல் முறை கடினத்தன்மை’ – பேரே புதிதாக இருக்கிறது – ஒன்றுமே புரியவில்லை – என்றெல்லாம் நினைக்கத் தொடங்கி விடாதீர்கள். இன்னும் அடுத்த ஐந்து நிமிடங்களுக்குள் புரிந்து விடும். கீழுள்ள படத்தைப் பாருங்கள்.

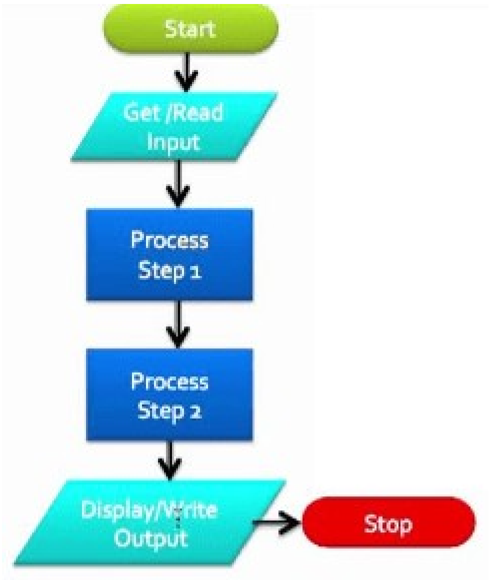


ஒரு நிரலைத் தான் இப்படிப் படமாக வரைந்து வைத்திருக்கிறோம்.

இங்கு மொத்தம் உள்ள மூடிய பகுதிகள் 5, வெளிப்பகுதி 1 என மொத்தப் பகுதிகள் 6.

சுழல்முறை கடினத்தன்மை என்பது எப்படிச் சுழன்றாலும் இந்த நிரலைச் சோதிப்பதற்கு என்னென்ன வழிமுறைகள் இருக்கின்றன என்று சொல்வது! அவ்வளவுதான்! மேல் உள்ள நிரலில் மொத்தப் பகுதிகள் 6. அதாவது, ஆக, இதன் கடினத்தன்மை சற்றுக் கூடுதல் தான்!

இதே போல் கீழ் உள்ள நிரலைப் பாருங்கள். ஒரே நேர்கோட்டில் இந்த நிரல் பயணிக்கிறது.



சுழல்முறை கடினத்தன்மை – கண்டுபிடிப்பது எப்படி?

முதல் நிரலில் உள்ள வட்டங்கள் 13. இந்த வட்டங்களை ‘Nodes’ என்று குறிப்பிடுவார்கள். மொத்தமுள்ள அம்புக்குறிகள் 17. அம்புக்குறிகள் ‘Edges’ எனப்படும். பிரிக்கும் வட்டங்கள் – அதாவது எந்தெந்த வட்டங்களில் இருந்து ஒன்றுக்கு மேற்பட்ட அம்புக்குறிகள் வருகின்றனவோ (அந்த அம்புக்குறிகள் – புது வழிகளைக் காட்ட வேண்டும்) அவை – பிரிக்கும் வட்டங்கள் (Predicate Nodes) எனச் சொல்லப்படுகின்றன. அவற்றின் எண்ணிக்கை இங்கு, 5.

இங்கு சுழல்முறை கடினத்தன்மை(Cyclomatic Complexity – CC) என்பது,

CC = E-N+2

CC -> Cyclomatic Complexity, E – Edges, N – Nodes

அதாவது, CC = 17-13+2 => 6. அதாவது 6 வழிகளில் இந்த நிரல் கட்டாயம் சோதிக்கப்பட வேண்டும். எவையெவை அந்த ஆறு வழிகள்?

- 1-2-3-5-7-11-13
- 1-2-3-5-8-11-13
- மேல் உள்ள 2 வழிகளில் 11 இல் இருந்து (படத்தைப் பார்க்கவும்) 2 க்குப் போய் மீண்டும் அதே வழியில் வருவது
- 1-2-4-6-9-12-13
- 1-2-4-6-10-12-13
- மேல் உள்ள 2 வழிகளில் 12 இல் இருந்து (படத்தைப் பார்க்கவும்) 4 க்குப் போய் மீண்டும் அதே வழியில் வருவது

சுழல்முறை கடினத்தன்மையை முடிவு செய்வதில் ஒரு வட்டதையோ, அம்புக்குறியையோ நீங்கள் எண்ணாமல் விட்டால் கூட, கடினத்தன்மையின் மதிப்பு மாறி விடும். எனவே, ஏதாவது ஒரு மாற்று வழியில் அந்தக் கடினத்தன்மை மதிப்பு சரியானது தான்! என்பதை உறுதிப்படுத்த வேண்டும் அல்லவா?

அந்தச் சூத்திரம் தான் –
CC = P + 2
P – Predicate Node – பிரிக்கும் வட்டம்
இங்கு பிரிக்கும் வட்டங்கள் – 2, 5, 4, 6 என மொத்தம் நான்கு.
எனவே CC = 4 + 2 => 6
இந்தக் கடினத் தன்மை மதிப்பும் இதற்கு முன் நாம் கண்டுபிடித்த மதிப்பும் பொருந்த வேண்டும். அப்போது தான், நாம் சரியான மதிப்பைக் கண்டுபிடித்திருக்கிறோம் என்று அர்த்தம்!
இப்போது மேல் உள்ள முதல் நிரலுக்கு, வரிவரிச் சோதனை, கிளைவரிச் சோதனை, வழிச்சோதனை ஆகியனவற்றின் மதிப்புகளைக் கொஞ்சம் யோசிக்கலாமா?
வழிச் சோதனை – 6 – இப்போது தான் பார்த்தோம் – எத்தனை வழிகள் இருக்கின்றன என்று!
வரிவரிச் சோதனை – இதற்கான எளிய வழிமுறை – எல்லா வட்டங்களையும் (Edges) இணைக்கும் வழிகள் எத்தனை இருக்கின்றன என்று பார்ப்பது – இங்கு

- 1-2-3-5-7-11-13
- 1-2-3-5-8-11-13
- 1-2-4-6-9-12-13
- 1-2-4-6-10-12-13

என்று நான்கு வழிகள் இருக்கின்றன. எனவே, வரிவரிச் சோதனையின் மதிப்பு 4 ஆகும்.
கிளைவழிச் சோதனை – இதற்கான எளிய வழிமுறை – எல்லா அம்புக்குறிகளையும் (Nodes) இணைக்கும் வழிகள் எத்தனை இருக்கின்றன என்று பார்ப்பது – இங்கு

- 1-2-3-5-7-11-13
- 1-2-3-5-8-11-13
- 1-2-4-6-9-12-13
- 1-2-4-6-10-12-13
- 1-2-3-5-7-11-2-மீண்டும் 13 வரை,
- இதே போல், 1-2-4-6-9-12-4-மீண்டும் 13 வரை

எனக் கிளை வழிச் சோதனையின் மதிப்பு 6 ஆகும்.

முதன்மையான குறிப்பு:

100 % வழிச் சோதனை செய்தால் 100% கிளைவழிச் சோதனை செய்தது போல் ஆகும்.
100% கிளைவழிச் சோதனை செய்வது 100% வரிவரிச் சோதனை செய்தது போல் ஆகும்.

மாற்றவழிச் சோதனையை(Mutation Testing) அடுத்த பதிவில் பேசுவோம்.

வெள்ளைப் பெட்டி உத்திகள் -4

மாற்ற வழிச் சோதனை(Mutation Testing)

அதென்ன மாற்ற வழிச் சோதனை? ஒரு சின்ன கதை வழியாக இதைப் புரிந்து கொள்வோம். அருள், வியன் – இருவரும் நண்பர்கள்; மென்பொறியாளர்கள். இருவரும் இணைந்து இணையத்தளம் ஒன்றை வடிவமைக்கிறார்கள். இணையத்தளத்தின் பின்னணி நிறம் சிவப்பாக இருந்தால் பளிச்சென்று எல்லோருக்கும் பிடித்தது போல் இருக்கும் என்று நினைக்கிறார் அருள். ஆனால், வியனுக்கோ வேறொரு எண்ணம் – பின்னணி நிறம் பச்சையாக இருந்தால், பார்ப்பதற்குப் பசுமையாக இருக்குமே என்பது வியனின் கருத்து. இருவரும் ஒருவரை இன்னொருவர் மாற்ற முயல்கிறார்கள் – முடியவில்லை. தங்கள் கருத்தை மற்றொருவர் மீது திணிக்க இருவருக்கும் விருப்பம் இல்லை. இருவரும் உட்கார்ந்து யோசிக்கிறார்கள். அப்போது அருள் ஒரு யோசனையைச் சொல்கிறார் – “இந்த இணையத்தளத்தை ஒரு வாரம் சிவப்புப் பின்னணியோடு வெளியிட்டுப் பார்ப்போம் – சிவப்புப் பின்னணி பிடித்திருக்கிறதா என்று கேட்டு ஒரு பொத்தானை இணையத்தளத்தில் வைத்து விடுவோம். எத்தனை பேர் விரும்புகிறார்கள் என்று கணக்கிட்டுக் கொள்வோம். பிறகு – ஒரு வாரம் கழித்து – தளத்தின் பின்னணியைப் பச்சை நிறத்திற்கு மாற்றி விடுவோம். திரும்பவும் இதே போல், பச்சைப் பின்னணியை எத்தனைப் பேர் விரும்பியிருக்கிறார்கள் என்று பார்ப்போம். நம்மை விட, வாடிக்கையாளர்கள் அறிவாளிகள் அல்லவா? அவர்களுடைய முடிவில் எது பெரும்பான்மையோ அதை எடுத்துக் கொள்வோம்!” – இது தான் அந்த யோசனை! யோசனையைக் கேட்ட உடனேயே வியனுக்குப் பிடித்துப் போய் விடுகிறது. “இந்த யோசனைப்படி, இன்னும் நீலம்,மஞ்சள் என்று எல்லா நிறங்களையும் கூட மாற்றிப் பார்க்கலாமே!” என்கிறார் வியன். “ஆமாம்! வேண்டுமானால் அதையும் செய்யலாம்” – இது அருள்.

இந்தக் கதை உங்களுக்குப் புரிந்தால் போதும். நீங்கள் மாற்ற வழிச் சோதனையின் அடிப்படையைப் புரிந்து கொண்டீர்கள். இதே தான் மாற்ற வழிச் சோதனை! மேல் உள்ள கதையில் இணையத்தளத்தை எடுத்து வேறு வண்ணம் பூசினார்கள் அல்லவா? அதற்குப் பதிலாக,

1) ஒரு மென்பொருள் நிரலை எடுத்துக் கொள்ள வேண்டும். அதில் சின்னச் சின்ன மாற்றங்கள் செய்து – ஒவ்வொரு மாற்றத்திற்கும் ஒரு நகல்(படி) எடுத்து வைத்துக் கொள்ள வேண்டும். இந்த நகலுக்குத் தான் மாற்ற வழி (Mutant) என்று பெயர்.

2) இப்போது ஏற்கெனவே நாம் எழுதி வைத்திருக்கும் டெஸ்ட் கேஸ்களை மாற்றங்கள் செய்யப்படாத மூல நிரலிலும் செயல்படுத்திப் பார்க்க வேண்டும்; மாற்றங்கள் செய்யப்பட்ட நகல்(படி)களிலும் செயல்படுத்திப் பார்க்க வேண்டும்.

3) பிறகு, டெஸ்ட் கேஸ்களின் முடிவுகள் அனைத்தையும் ஒன்றாகத் தொகுத்து ஒப்பிட வேண்டும்.

4) இதில் எந்த மாற்ற வழி – குறைவான குறைகளையோ அல்லது குறைகளே இல்லாத வழியையோ காட்டுகிறதோ – அதை இறுதி நிரலாக எடுத்துக் கொள்ளலாம்.

இப்படிச் செய்வது தான் மாற்ற வழிச் சோதனை (Mutation Testing) என்பதாகும். இந்தச் சோதனையை எளிதாக்க, கட்டற்ற மென்பொருள் ஏதாவது இருக்கிறதா? என்று கேட்கிறீர்களா? இல்லாமல் இருக்குமா? ஜம்பிள்(Jumble), ஸ்டிரைக்கர்(Stryker) என்று நிறைய இருக்கின்றன.

இந்தப் பதிவோடு வெள்ளைப் பெட்டிச் சோதனை முறைகளை மூட்டை கட்டி விடலாம். இனி அடுத்த பதிவில் கருப்புப் பெட்டி, வெள்ளைப் பெட்டிச் சோதனைகளில் கண்டுபிடிக்கப்படும் பிழைகளை எப்படிப் பதிவது? யாரிடம் சொல்வது? அந்தப் பிழைகளைத் திருத்த வேண்டியது யார்? இப்படிப் பல கேள்விகளைப் பார்க்க வேண்டியது இருக்கிறது – தொடர்ந்து பேசுவோம்!

பிழை கண்டுபிடிப்பது - பிழைப்பே அது தான்!

இது பிழைகளைப் பற்றிப் பேச வேண்டிய நேரம். இப்போது வரை, டெஸ்ட் கேஸ்கள் எழுதுவது, உத்திகள் வகுத்து சோதிப்பது - ஆகியவற்றைப் பார்த்து விட்டோம். இப்போது நாம் கண்டுபிடிக்கும் பிழைகளை - எங்கே பதிவது? யாரிடம் சொல்வது? யார் அதைப் பார்ப்பார்கள்? யார் திருத்துவார்கள்? அவர்கள் திருத்தியது, நமக்கு எப்படித் தெரிய வரும்? அதன் பிறகு டெஸ்டர்களாகிய நாம் செய்ய வேண்டியது என்ன? இப்படி அடுக்கடுக்காக நிறைய கேள்விகள் இருக்கின்றன. நிறைய கேள்விகள் என்றால் - பதில்களும் நிறைய இருக்குமே என்று நினைக்கிறீர்களா? அது தான் இல்லை! மிக எளிதாகப் பார்த்து விடலாம். பிழைகளைப் பற்றிப் பேசுவதற்கு முன்பு, வேறொரு கதை பேசுவோம். கதை தான் நமக்குப் பிடிக்குமே! இந்த முறை நம்முடைய நாயகனின் பெயர் கவின். கவினுக்குச் சின்ன வயதில் இருந்தே அலைபேசிகள் மீது அலாதி ஆர்வம். அவனுக்குக் கல்லூரிக் காலத்திலேயே எல்லாவித அலைபேசிகள் பற்றி அத்துப்படி! படிப்பு முடித்து வேலைக்கு வந்த பிறகு அலைபேசி தான் அவனுக்கு உற்ற நண்பன் என்று சொல்லும் அளவு அலைபேசிப் பிரியன். அப்படிப்பட்டவனுக்கு வந்தது சோதனை!

கடந்த சில நாட்களாக எந்த அழைப்பும் அவனுக்கு வருவதுமில்லை; அவனால் யாரையும் அழைக்கவும் முடியவில்லை. அவனும் அலைபேசியை அணைத்து உயிர்ப்பது, சிம் அட்டையைக் கழற்றி மாட்டுவது என்று எல்லா வகை முயற்சிகளையும் செய்து விட்டான். சரி செய்தபாடில்லை. கடைசியாக என்ன செய்வதென்றே தெரியாமல், சிம் நிறுவனத்தின் வாடிக்கையாளர் சேவை மையத்தைத் தொடர்பு கொண்டான். தமிழுக்கு எண் ஒன்றை அழுத்தினான். எதிர்முனையில் ஓர் இனிமையான குரல் - 'வணக்கம்! தங்களுக்காக உதவக் காத்திருப்பது வெண்பா! என்ன சிக்கல்னு தெரிஞ்சிக்கிடலாமா?". "வணக்கங்க! என் பெயர் கவின். கடைசி 2,3 நாளாவே என் பேசியில் இருந்து யாருக்கும் அழைப்பு போகல, எனக்கும் யாருடைய அழைப்பும் வரல. " என்றான் கவின். "உங்க கஷ்டம் எங்களுக்குப் புரியுது! நீங்க என்ன அலைபேசி பயன்படுத்துறீங்கனு தெரிஞ்சுக்கலாமா?" - வெண்பா. கவின் சொன்னான். "அந்தப் பேசியில, ஸ்லாட் 1 ஆ, ஸ்லாட் 2 ஆ - எந்த ஸ்லாட்ல - நீங்க

சிம் அட்டையை வச்சிருக்கீங்கன்னு கொஞ்சம் சொல்ல முடியுமா சார்?" - இந்தக் கேள்வி கவினைச் சற்றே எரிச்சல் படுத்தினாலும் கட்டுப்படுத்திக் கொண்டு, "2 ஸ்லாட்ல எந்த ஸ்லாட்டில் இருந்தாலும் சிம் வேலை செய்யனும் இல்லையா?" என்று கேட்டான். இதன் பிறகு, வெண்பா தரப்பில் இருந்து, "சிம்மை ஒரு தடவை கழற்றி மாட்டினீர்களா? அலைபேசியை ஒரு தடவை அணைத்து உயிர்ப்பித்தீர்களா?" என்று பாரம்பரியமாக வாடிக்கையாளர் சேவை மையத்தில் இருந்து கேட்கப்படும் கேள்விகள். எல்லாக் கேள்விகளுக்கும் பொறுமையாகப் பதில் சொன்னான் கவின். கடைசியில் "எங்களுடைய தொழில்நுட்ப அணியில் இருந்து இன்னும் 24 மணி நேரத்திற்குள் உங்களைத் தொடர்பு கொள்வார்கள். இந்த நாள் இனிய நாளாக அமைய எங்களுடைய வாழ்த்துகள்" என்றவாறு தொடர்பைத் துண்டித்தாள் வெண்பா.

இப்போது கதையைப் படித்துக் கொண்டிருக்கும் உங்களிடம் ஒரு கேள்வி - வாடிக்கையாளரிடம் பேசத் தொடங்கியதில் இருந்து தொடர்பைத் துண்டித்தது வரை வெண்பா என்னென்ன செய்திருப்பாள்?

1) அழைப்பு வந்த உடன், என்ன சிக்கல் என்று கேட்டிருக்க வேண்டும். அதைக் குறித்து வைத்திருக்க வேண்டும்.

2) சிக்கல் பற்றி விவரமாகத் தெரிந்து கொண்டு அதையும் குறித்து வைத்திருக்க வேண்டும்.

3) வாடிக்கையாளர் சிக்கலைத் தீர்க்க என்னென்ன செய்தார் - அப்படியும் சிக்கல் தீரவில்லை என்பதைக் கேட்டு குறித்து வைத்திருக்க வேண்டும்.

4) வாடிக்கையாளரின் சிக்கலின் தீவிரம் எப்படிப்பட்டது எனத் தெரிந்து கொண்டிருக்க வேண்டும். அதைப் பொறுத்துத் தான், சிக்கலை 24 மணிநேரத்தில் தீர்ப்பதா? 48 மணிநேரத்தில் தீர்ப்பதா? இல்லை இன்னும் அதிக நேரம் எடுக்கலாமா? என்று முடிவு செய்திருக்க வேண்டும்.

5) மேற்கொண்டு இந்தச் சிக்கலைத் தீர்க்கும் வேலையை யாருக்கு ஒதுக்குவது என்று முடிவு செய்திருக்க வேண்டும்.

6) தான் வாடிக்கையாளரிடம் என்ன பேசினோம் என்பதைக் குறித்து வைத்திருக்க வேண்டும்.

7) இவை தவிர்த்து, வாடிக்கையாளர் - சேவை மையத்தைத் தொடர்பு கொண்ட நாள், நேரம், அந்த அழைப்பை எடுத்தவர் யார், அந்த அழைப்புக்கென தனியே ஓர் எண், அழைப்பு முடிந்ததும் தானியங்கியாக வாடிக்கையாளர் எண்ணுக்கு ஒரு குறுஞ்செய்தி - ஆகிய அனைத்தையும் வாடிக்கையாளர் சேவை மைய நிறுவனம் ஒருங்கிணைத்திருக்க வேண்டும்.

இவை தாமே உங்களுடைய ஊகங்கள்! செம்மையாகச் சொன்னீர்கள் போங்கள். இதே போல, ஒரு கதையைத் தான் டெஸ்டரும் சொல்லப் போகிறார் - ஒரே ஒரு வித்தியாசத்தோடு! அந்த ஒரே ஒரு வித்தியாசம் என்ன என்று கேட்கிறீர்களா? சாதாரணமாக, வாடிக்கையாளர் சேவை மையத்தில் - ஓர் அழைப்பில் - வாடிக்கையாளர், சேவை மைய அதிகாரி என இருவர் உண்டு. மென்பொருள் சோதனையில் ஒரே ஒருவர் மட்டும் தான்! அவர் தாம் டெஸ்டர்!

1) டெஸ்டரே வாடிக்கையாளராகத் தன்னை நினைத்துக் கொண்டு தவறுகளைக் கண்டுபிடிக்க வேண்டும். 2) அவரே வா. சே. மைய அதிகாரியாகத் தன்னை உருவகித்துக் கொண்டு - அந்தத் தவறுகளைப் பதிய வேண்டும். இரண்டையும் ஒருவரே செய்ய வேண்டும்.

தவறுகள் (அல்லது பிழைகள்) கண்டுபிடித்து அதைப் பதிந்து உரிய அணிக்குத் தெரியப்படுத்த வேண்டும். உரிய அணி எது என்பது தெரியாத சூழலில் தன்னுடைய அணித்தலைமைக்குத் தெரியப்படுத்த வேண்டும். நாம் மேலே பார்த்த கவின் - வெண்பா உரையாடலையே ஒரு பிழையாகப் பதிந்தால் எப்படி இருக்கும் என்று பார்ப்போமா?

பிழை எண்: 1234 நாள்: 27-அக்டோபர்-2019 நேரம்: 07:45 காலை
கண்டுபிடித்தவர்: கவின் தீர்ப்பது: வெண்பா
நிலை: புதிது
தலைப்பு: இரண்டு நாட்களாக அழைப்பு வர/போகவில்லை.
சிக்கலின் பாதிப்பு: தீவிரம் தீர்ப்பது: எளிதாகத் தெரியவில்லை
விவரம்:
கடந்த 2, 3 நாட்களாக, வாடிக்கையாளருக்கு எந்த அழைப்பும் வரவில்லை. அவராலும் யாரையும் கூப்பிட முடியவில்லை. ஆனால், குறுஞ்செய்தி அனுப்ப/ பெற முடிகிறது.

படிகள்:
1) வாடிக்கையாளருக்கு கடந்த 2, 3 நாட்களாக, வாடிக்கையாளருக்கு எந்த அழைப்பும் வரவில்லை. அவராலும் யாரையும் கூப்பிட முடியவில்லை.
2) அவர் தம்முடைய சிம்மை ஸ்லாட் மாற்றி வைத்து முயன்றிருக்கிறார். அப்படியும் சிக்கல் தீரவில்லை.
3) ஓரிரு முறை, அலைபேசியை அணைத்து உயிர்ப்பித்திருக்கிறார். அப்படியும் சிக்கல் தீரவில்லை.
4) வாடிக்கையாளரின் கணக்கில் போதுமான அளவு இருப்பும் இருக்கிறது.
இயங்கு சூழல்: சாம்சங் ஜே7 புரோ, ஆண்டிராய்டு 6
சான்றுகள்:
அலைபேசியின் ஸ்கிரீன்ஷாட், அழைப்பை முயலும் போது சிம் நிறுவனத்தில் உருவாகும் அடிப்படையான சில கோப்புகள்

இந்தத் தகவல்களைத் தாம் - பிழையைப் பதியும் போது ஒரு டெஸ்டர் சொல்ல வேண்டும். இதைத் தான் பிழை பதித்தல் (Bug Reporting) என்று சொல்வார்கள். இதில், சிக்கலின் பாதிப்பு, தீர்ப்பது என்றெல்லாம் என்னென்னவோ சொல்கிறீர்களே அவையெல்லாம் என்ன? இந்த எடுத்துக்காட்டில் கண்டுபிடித்தவர் கவின் என்றும் தீர்ப்பது: வெண்பா என்றும் சொல்லியிருக்கிறீர்கள். மென்பொருள் சோதனையில் இந்த இரண்டுமே டெஸ்டரைத் தானே குறிக்கும்? நிலை: புதிது என்றால் என்ன? இயங்கு சூழல், சான்றுகள் எதற்காகச் சேர்த்திருக்கிறீர்கள்? இப்படி அடுக்கடுக்காக நிறைய கேள்விகள் வருகின்றனவா? இந்தக் கேள்விகளுக்கும் இவற்றோடு சேர்த்து பிழை வாழ்க்கை வட்டம் என்றால் என்ன என்பதையும் அடுத்த பதிவில் பார்ப்போமா?

‘பிழை’ப்பைத் தொடர்வோம்!

போன பதிவு பல கேள்விகளுடன் முடிந்திருந்தது. அந்தக் கேள்விகளுக்கும் எல்லாவற்றிற்கும் பதில் பார்த்து விடுவோமா?

பிழை எண் (Bug ID):

ஒவ்வொரு பிழைக்கும் ஒதுக்கப்படும் தானியங்கி எண். இந்த எண்ணைக் கொண்டு தான் பிழையை அடையாளம் கண்டுபிடிப்பார்கள்.

கண்டுபிடித்தவர் (Opened By):

பிழையைக் கண்டுபிடித்தவர் பெயரைப் பதிவதற்காக இந்த ஏற்பாடு. பிழையில் ஏதாவது சந்தேகம் வந்தால், இவரைக் கேட்கலாம் அல்லவா? அதற்காகத் தான்!

பொறுப்பு (Assigned To):

பிழையைச் சரி செய்வது யார்? என்று இங்குக் குறிப்பிடுவார்கள். யார் சரி செய்வது என்று முடிவு செய்து விட்டால், அந்த நிரலர் பெயரை நேரடியாகச் சொல்லி விடலாம். இல்லையெனில் எந்த அணி - பிழையைத் தீர்க்க வேண்டிய பொறுப்பில் உள்ளதோ அந்த அணித் தலைவர் பெயரைக் குறிப்பிடலாம்.

நிலை (Status):

பிழையின் தற்போதைய நிலை என்ன என்று சொல்வது. இதில் புதியது, ஒப்படைக்கப்பட்டது, தீர்க்கப்பட்டது, முடிந்தது எனப் பல நிலைகள் இருக்கலாம். இதைப் பற்றி விரிவாகப் பின்னர் பார்ப்போம்.

தலைப்பு (Title):

பிழையின் தலைப்பு - பிழை எதைப் பற்றியது என்று சொல்வது.

சிக்கலின் பாதிப்பு(Severity):

இப்போது நாம் பதியும் பிழை - வாடிக்கையாளருக்கு எவ்வளவு பாதிப்பை ஏற்படுத்தும் என்று சொல்வது. ஒரு டெஸ்டருக்கு - வாடிக்கையாளருக்கு ஏற்படும் பாதிப்பு பற்றிய பட்டறிவு தேவை என்பதால், இதை டெஸ்டரே பதிய வேண்டும். பொது நிலையில், பாதிப்பை

- குறைவு

- *நடுத்தரம்*
- *அதிகம்*

என்று வகைப்படுத்துவார்கள். சில நேரங்களில் 1,2,3 *என எண்களைக் கொண்டு வரிசைப்படுத்துவதும் உண்டு.*

***தீர்வு*(Priority):**

ஒரு தொழில்நுட்பச் சிக்கலைத் தீர்ப்பதற்கு எவ்வளவு காலம் ஆகும் என்பதை ஒரு டெஸ்டரால் சொல்ல முடியாது அல்லவா? அதனால் இதை மட்டும் டெஸ்டர் நிரப்பாமல் விட்டு விடுவார். தொழில்நுட்ப அணியோ, நிரலர்(developer) *அணியோ இதை,*

- *எளிது*
- *நடுத்தரம்*
- *கடினம்*

என்று வகைப்படுத்துவார்கள்.

சுருக்கமாகப் புரிந்து கொள்வதாக இருந்தால்:

- *பாதிப்பு - பயனருக்கு ஏற்படும் பாதிப்பைப் பொருத்தது.*
- *தீர்வு - தீர்வுக்கு ஆகும் நேரத்தைப் பொருத்தது.*

***விவரம்* (Description):**

தலைப்பைப் பற்றி ஓரிரு வரிகளில் விவரமாகச் சொல்வது.

***படிகள்*(Steps):**

என்னென்ன படிகளைச் செய்யும் போது இப்படிப்பட்ட பிழை வந்தது என்பதைச் சொல்வது. இந்தப் படிகளைத் தான் நிரலர் அணி ஒவ்வொரு படியாகச் செய்து பார்த்து பிழையை உறுதிப்படுத்துவார்கள். அதற்காகத் தான் படிகளை வரிசைப்படிச் சொல்வது.

***இயங்குசூழல்* (Environment):**

பிழையை எந்தச் சூழலில் கண்டுபிடித்தோம் என்று சொல்வது - அதாவது, வன்பொருள் தகவல்கள்(இன்டெல் ஐ 3 *என்பது போன்று), மென்பொருள் தகவல்கள் (லினக்ஸ் மின்ட்* 19, 64 *பிட் இயங்கு தளம், பயர்பாக்ஸ் உலாவி என்பன போன்று) ஆகியவற்றைச் சொல்வது. சில சமயங்களில் ஒரு பிழை - எல்லா இயங்குசூழல்களிலும் வராது. சில சூழல்களில் மட்டும் வரும். அது போன்ற சமயத்தில் பிழையைப் போக்கும் நிரலருக்கு இயங்குசூழல் பற்றிய விவரங்கள் இன்றியமையாதவை அல்லவா? அதற்குத் தான் இந்த ஏற்பாடு.*

இணைப்புகள்(Attachments):

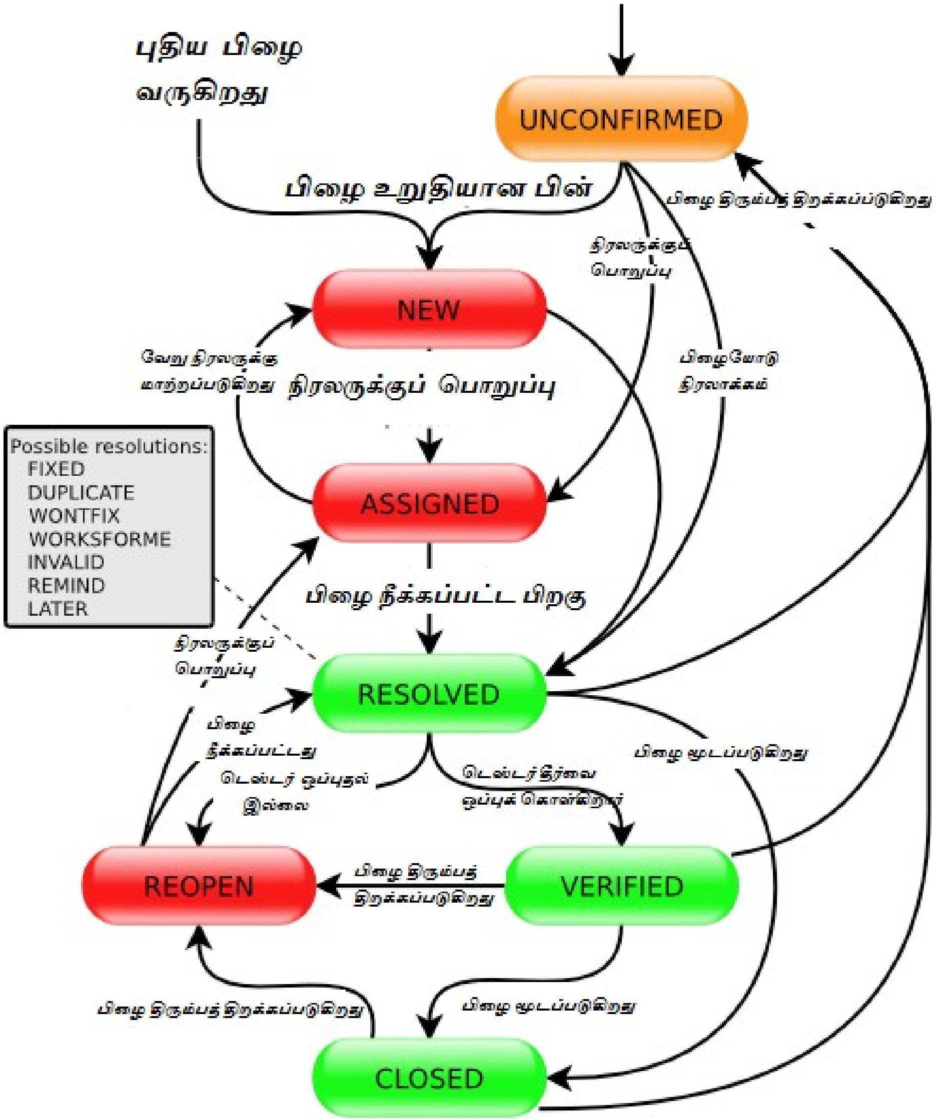
டெஸ்டர் கண்டுபிடித்த பிழைக்குச் சான்றாக இருக்கும் அனைத்து விவரங்களும் இதில் கொடுக்கப்படும். ஸ்கிரீன்ஷாட் (திரைப் பிடிப்பு), பிற கோப்புகள் ஆகியன இதில் சேர்க்கப்படும்.

பிழை பதிப்பது (Bug Reporting) *எப்படி என்பதைத் தான் இப்போது நாம் பார்த்திருக்கிறோம். இன்னும் பிழை வாழ்க்கை வட்டத்தைப் பற்றி நாம் பேசவே இல்லையே!* என்கிறீர்களா? *அடுத்த பதிவில் அந்த வட்டத்தை வரைந்து விடுவோம்.*

பிழை வாழ்க்கை வட்டம்(Bug Life Cycle)

'வாழ்க்கை ஒரு வட்டம்' என்று தெரியும் - அதென்ன பிழை வாழ்க்கை வட்டம்? பிழையான வாழ்க்கை வட்டமா? என்று கேட்கிறீர்களா? இல்லை! நம்முடைய வாழ்க்கை எப்படி ஒரு வட்டமோ, அதே போல, சாப்ட்வேர் டெஸ்டிங் மூலமாக நாம் மென்பொருளில் கண்டுபிடிக்கும் பிழைகளுக்கும் ஒரு வாழ்க்கை வட்டம் இருக்கிறது! எனவே, இது பிழையின் வாழ்க்கை வட்டம்! அதைப் பற்றிப் பார்ப்போம்!

மென்பொருள் உருவாக்கத்திற்குப் பிறகு, டெஸ்டர்கள் மென்பொருளைச் சோதிக்கிறார்கள். அந்தச் சோதனை மூலம் பல்வேறு பிழைகளைக் கண்டுபிடிக்கிறார்கள். அந்தப் பிழைகள் கடந்து வரும் பல்வேறு நிலைகள் தாம் பிழை வாழ்க்கை வட்டத்தை அமைக்கின்றன. அந்த நிலைகள் என்னென்ன? ஏன் பல நிலைகள் தேவைப்படும்? பார்ப்போமா?



(பட உதவி: https://commons.wikimedia.org/wiki/File:Bugzilla_Lifecycle_color-aqua.png)
நம் வாழ்க்கை எப்படிப் பல நேரங்களில் பல வட்டம் அடிக்குமோ, அதே போல் தான் பிழை வாழ்க்கை வட்டமும்! ஒரே நேர்கோட்டில் பயணிக்காது. ஒவ்வொரு நேரமும் ஒவ்வொரு கோட்டில் பயணிக்கும். ஒவ்வொன்றாகப் பார்ப்போமா?

முதல் கோடு:

UNCONFIRMED → NEW → ASSIGNED → RESOLVED → VERIFIED → CLOSED
1) எப்போதுமே பிழை கண்டுபிடிக்கப்படும் போது உறுதிப்படுத்தப்படாத நிலை(UNCONFIRMED)யில் இருக்கும்.
2) பிற டெஸ்டர்கள் அந்தப் பிழையை உறுதிப்படுத்தும் போது, நிலை புதியது(NEW) என மாற்றப்படும்.
3) பிறகு அந்தப் பிழை - நிரலர் அணியில் உள்ள ஏதாவது ஒரு நிரலருக்கு ஒதுக்கப்படும். அப்போது நிலை பொறுப்பு(ASSIGNED) என மாற்றப்படும்.
4) நிரலர் பிழையை நீக்கிய பிறகு, அவர் பிழையின் நிலையைத் தீர்ந்தது(RESOLVED) நிலைக்கு மாற்றி விடுவார்.
5) பிறகு அது டெஸ்டருக்கு ஒதுக்கப்படும். டெஸ்டர் சோதித்துப் பார்த்து - பிழை நீக்கப்பட்டிருந்தால் சரிபார்க்கப்பட்டது(VERIFIED) என மாற்றப்பட்டு கடைசியாக 'முடிந்தது'(CLOSED) நிலைக்கு மாற்றப்படும்.

இரண்டாவது கோடு:

UNCONFIRMED → NEW → ASSIGNED → RESOLVED → REOPENED → ASSIGNED → RESOLVED →CLOSED
முதல் கோட்டிற்கும் இரண்டாவது கோட்டிற்கும் உள்ள வித்தியாசம் - மூன்றாவது படியில் நிரலர் பிழையை நீக்கி விட்டதாகச் சொல்கிறார். ஆனால் டெஸ்டர் - தம்முடைய சோதனையில் பிழை நீங்கவில்லை என உறுதி செய்கிறார். அதாவது, தீர்ந்தது → மறுதிறப்புக்குப் போய் மீண்டும் தீர்கிறது.

மூன்றாவது கோடு:

சில சமயங்களில் பல டெஸ்டர்கள் ஒருவர் பிழை பதிந்ததை அறியாமல் மற்றவர்களும் அதே போலப் பிழையைப் பதிந்து விடுவார்கள். இது போன்ற சூழலில், அந்தப் பிழைகள் போலி எனப் பதியப்பட்டு நீக்கப்பட்டு விடும். அதாவது, அந்தக் கோடு
UNCONFIRMED → NEW → ASSIGNED → DUPLICATE → RESOLVED → VERIFIED → CLOSED
என அமையும்.

நான்காவது கோடு:

வேறு சில சமயங்களில் டெஸ்டர் பதியும் பிழை, உண்மையான பிழையாக இல்லாமல், அவருடைய கணினியில் மட்டும் சரிவர இயங்காமல் இருக்கும். அதைப் பிழை என நினைத்து அவரும் பதிந்து விடுவார். அப்படிப்பட்ட பிழைகள், சூழல் காரணமாக அமைந்த பிழைகள் (Environmental Issues) என அறியப்பட்டு நீக்கப்படும். அந்தக் கோடு,
UNCONFIRMED → NEW → ASSIGNED →WORKS FOR ME → RESOLVED → VERIFIED → CLOSED என்னும் வகையில் இருக்கும்.

ஐந்தாவது கோடு:

வாடிக்கையாளருடைய தேவைகள் அனைத்தும் தேவை சுவட்டு ஆவணத்தில் தெளிவாகக் கொடுக்கப்பட்டிருக்கும். அந்த ஆவணத்தைச் சரிவர புரிந்து கொள்ளாமல் டெஸ்டர்கள் சில நேரங்களில் பிழைகளைப் பதியும் நிலை உண்டு. அந்த நேரங்களில், அவை தகுதியற்றவை(INVALID) என நிலை மாற்றப்பட்டு - நீக்கப்படும்.
UNCONFIRMED → NEW → ASSIGNED →INVALID → RESOLVED → VERIFIED → CLOSED

ஆறாவது கோடு:

டெஸ்டர்கள் உருவாக்கும் பிழைகளின் பாதிப்பு (Severity) குறைவாக இருக்கும் போது, நிரலர்களுக்கு இருக்கும் பணிச்சுமையைப் பொறுத்து சில பிழைகள் உறுதிப்படுத்தப்பட்டாலும் பின்னர்(LATER) அல்லது நினைவூட்டு(REMIND) நிலைக்கு மாற்றப்படும்.
UNCONFIRMED → NEW → ASSIGNED →LATER → RESOLVED → VERIFIED → CLOSED
UNCONFIRMED → NEW → ASSIGNED →REMIND → RESOLVED → VERIFIED → CLOSED

ஏழாவது கோடு:

நிரலர்களுக்குப் பிழைகளை நீக்கும் வேலை ஒதுக்கப்பட்ட பிறகு சில சூழல்களில் வேறு நிரலருக்கு அந்தப் பிழை ஒதுக்கப்படும் நிலை வரலாம். அதாவது,
UNCONFIRMED → NEW → ASSIGNED →NEW → RESOLVED → VERIFIED → CLOSED
இவை போல இன்னும் சில கோடுகள் கூட நம்மால் யோசிக்க முடியும். ஆனால், பொது நிலையில் இருக்கும் பிழை வாழ்க்கை வட்டம் இவ்வளவு தான்! இப்பொழுது - பிழை வாழ்க்கை வட்டம் (Bug Life Cycle) பற்றி ஓரளவு உங்களுக்குப் புரிந்திருக்கும் என்று நம்புகிறேன்.

மென்பொருள் சோதனை வகைகள்

பொதுவாக மென்பொருள் சோதனைகளை(Software Testing - Types) இரண்டு வகைகளாகப் பிரிக்கலாம்.

- நிலைத்த வகை சோதனை (Static Testing)
- இயக்க வகை சோதனை(Dynamic Testing)

நிலைத்த வகை சோதனை (Static Testing):

நிலைத்த வகை சோதனை என்பது உண்மையில் மென்பொருளைச் சோதிப்பது அன்று! மென்பொருளின் நிரல்(Code), தேவை ஆவணங்கள்(Requirement Documents), வடிவமைப்பு ஆவணங்கள்(Design Documents) ஆகியவற்றைச் சோதிப்பது ஆகும். மென்பொருளைச் சோதிப்பது என்பது மென்பொருளின் பயனைப் பொருத்து மாறும். ஆனால், நிரல், ஆவணங்கள் ஆகியவற்றைச் சோதிப்பது என்பது நிலையானது அல்லவா? அதனால் தான் இவ்வகைச் சோதனைக்கு நிலைத்த வகைச் சோதனை என்று பெயரிட்டிருக்கிறார்கள்.

நிலைத்த வகை சோதனைகளை மேலும் அதன் உட்பிரிவுகளாக,

=> திறனாய்வு முறை (Review)

=> விளக்க முறை (Walkthrough)

என்று இரண்டு வகைகளாகப் பிரிக்கலாம்.

திறனாய்வு முறை (Review) :

இந்தத் திறனாய்வை முறைசாராத் திறனாய்வாகவும் (Informal) செய்யலாம்; முறையான திறனாய்வாகவும்(Formal) செய்யலாம். முறைசாரா(Informal Review)த் திறனாய்வு என்பதில் வேறு ஒரு நிரலர் நிரல், ஆவணங்கள் ஆகியவற்றை நேரம் கிடைக்கும் போது ஆய்ந்து பார்ப்பார். அவர் தம்முடைய ஆய்வு முடிவாக என்னென்ன சொல்ல வேண்டும் என்பதற்கு எந்த வரையறையும் கிடையாது. தம்முடைய அனுபவத்தின் அடிப்படையில் அவர் திறனாய்வு செய்து முடிவுகளை விளக்குவார். இந்தத் திறனாய்வு முறைக்குத் தான் விளக்க முறை(Walkthrough) என்று பெயர்.

முறையான திறனாய்வு(Formal Review):

முறையான திறனாய்வு என்பது ஆய்வு(Inspection) என்று சொல்லப்படும். இந்தத் திறனாய்வில் யார் யார் கலந்து கொள்ள வேண்டும், எப்போது திறனாய்வுக் கூட்டம் நடக்கும், எங்கு நடக்கும் என்பன போன்ற எல்லாமே முறையாகத் திட்டமிட்டு நடத்தப்பட வேண்டும்.

- இந்த வகை திறனாய்வுக் கூட்டத்தை ஏற்பாடு செய்பவருக்கு ஏற்பாட்டாளர்(Moderator) என்று பெயர்.
- திறனாய்வுக் கூட்டத்தில் நடக்கும் அனைத்தையும் குறிப்பெடுப்பவருக்கு குறிப்பெடுப்பவர்(Scribe) என்று பெயர்.
- கூட்டத்தில் கலந்துகொண்டு திறனாய்வு செய்பவர்கள் திறனாய்வாளர்கள்(Reviewers) என்று சொல்லப்படுவார்கள். அவர்கள் எந்தெந்த நிரல்கள், ஆவணங்கள் ஆகியவற்றைத் திறனாய்வுக்கு உட்படுத்துவார்கள் என்பது பற்றிய பட்டியல் திறனாய்வுக்கூட்டத்திற்கு முன்னரே முறையா எல்லோருக்கும் கொடுக்கப்படும்.
- யாருடைய நிரல், ஆவணங்கள் ஆகியவற்றைத் திறனாய்வுக்கு உட்படுத்துகிறார்களோ - அவரை எழுத்தர்(Author) என்று சொல்வார்கள்.

இயக்க வகை சோதனை(Dynamic Testing)

இயக்க வகை சோதனைகளை நாம் முன்னரே விரிவாகப் பார்த்திருக்கிறோம். முழுமையான சோதனைக்குப் பிறகு வாடிக்கையாளர் செய்யும் சில சோதனைகள் இருக்கின்றன. என்ன – டெஸ்டர்கள் தாமே சோதிப்பார்கள்? வாடிக்கையாளர்களும் சோதிப்பார்களா? என்கிறீர்களா? எல்லாவகைச் சோதனைகளும் முழுமையாக முடிந்த பிறகு - வாடிக்கையாளர்களை ஈடுபடுத்தி வாடிக்கையாளர் ஏற்புச் சோதனைகள்(User Acceptance Testing) செய்யப்படும்.

  - ஆல்பா சோதனை (Alpha Testing)
  - பீட்டா சோதனை (Beta Testing)

ஆகிய சோதனைகள் செய்யப்படும். வாடிக்கையாளர்கள் என்றால் எல்லா வாடிக்கையாளர்களும் அல்லர்! தேர்ந்தெடுக்கப்பட்ட ஒரு சில வாடிக்கையாளர்கள் இந்தச் சோதனை வாய்ப்பு கொடுக்கப்படும். அதென்ன ஆல்பா சோதனை, பீட்டா சோதனை? பார்ப்போம் ஒவ்வொன்றாக!

ஆல்பா சோதனை:

நாம் வெளியிடப் போகும் மென்பொருளை முழுமையாக டெஸ்டர்கள் கொண்டு சோதித்த பிறகு - வெளியீட்டிற்கு முன் தேர்ந்தெடுக்கப்பட்ட சில வாடிக்கையாளர்கள் - நம் இடத்திற்கு வரவழைக்கப்படுவார்கள். வெளியிடப் போகும் மென்பொருளை நம் கணினிகளில் நிறுவி அவர்களிடம் கொடுப்போம். அவர்களை புதிய வெளியீட்டைப் பயன்படுத்திப் பார்க்க வேண்டுவார்கள். அவர்கள் பயன்படுத்திப் பிழைகள் ஏதாவது இருந்தால் சொல்வார்கள். இப்படிச் செய்வதில் ஒரே கல்லில் இரண்டு மாங்காய்!
* மென்பொருள் நிறுவனத்திற்கும் வாடிக்கையாளர் என்ன விரும்புகிறார் என்பது வெளியீட்டிற்கு முன்னர் தெரிந்து விடும்.

* தேர்ந்தெடுக்கப்பட்ட வாடிக்கையாளர்களும் இப்படி வெளியீட்டிற்கு முன்னரே (பிறர் பயன்படுத்தத் தொடங்குவதற்கு முன்னரே) ஒரு மென்பொருளைப் பயன்படுத்துவதைப் பெருமையாக நினைப்பார்கள்.

பீட்டா சோதனை:

பீட்டா சோதனை என்பதும் வாடிக்கையாளர் செய்வது தான்! ஆனால் ஒரே ஒரு வித்தியாசம்! ஆல்பா சோதனையை வாடிக்கையாளர் - மென்பொருள் நிறுவனத்தின் கணினிகளில் செய்து பார்ப்பார். பீட்டா சோதனையை வாடிக்கையாளர் தம்முடைய இடத்தில் இருந்து கொண்டே தம்முடைய கணினியிலேயே சோதித்துப் பார்ப்பது ஆகும். இது வரை நாம் சோதனை வகைகள் என்னென்ன என்பதைப் பார்த்து விட்டோம். மென்பொருள் சோதனைகள் செய்வதற்கு என்றே சில வழிகாட்டு நெறிமுறைகள்(Software Testing Principles) இருக்கின்றன. அவை என்னென்ன? தொடர்ந்து பார்ப்போம்.

மென்பொருள் சோதனை நெறிமுறைகள்

மென்பொருள் சோதனைக்கு அடிப்படையான வழிமுறைகளை ஏழு நெறிமுறைகளாக(Software Testing Principles)த் தொகுத்திருக்கிறார்கள். அவற்றைத் தாம் இந்தப் பதிவில் பார்க்கப் போகிறோம்.

நெறிமுறை #1:

பிழைகள் எங்கெங்கு இருக்கின்றன என்று காட்டுவது தான் சோதனை.
(Testing shows presence of defects.)
பிழைகள் எங்கெங்கு இருக்கின்றன என்று காட்டுவது தான் சோதிப்பது ஆகும். எனவே இந்த நெறிமுறையை மனத்தில் கொண்டு அதற்கேற்ப டெஸ்ட் கேஸ்களை உருவாக்க வேண்டும். அதற்காகப் பிழைகளே இல்லாத ஒரு மென்பொருளை உருவாக்கி விட முடியும் என்று நம்பாதீர்கள். (அப்படி ஒரு மென்பொருள் இருந்தால் எதற்கு அடுத்தடுத்த பதிப்புகள்(வெர்ஷன்கள்) தேவைப்படும்?)

நெறிமுறை #2:

எல்லாவகைச் சோதனைகளையும் செய்து முடிப்பது சாத்தியமே இல்லாத ஒன்று.
(Exhaustive Testing is impossible.).
ஒரு மென்பொருளைச் சோதிக்கும் போது எல்லாவகைச் சோதனைகளையும் சோதிப்பது என்பது இயலாத ஒன்றாகும். ஒரு கடவுச்சொல் பதியும் பெட்டியைச் சோதிப்பதாக வைத்துக் கொள்ளுங்கள். ஒவ்வொரு பயனரும், ஒவ்வொரு விதமாகத் தம்முடைய கடவுச்சொல்லை வைத்திருப்பார். அதை ஒவ்வொன்றையும் ஒரு டெஸ்டரால் உருவாக்கிச் சோதிக்க முடியாதல்லவா? ஒரு பொது வரையறையை உருவாக்கி அதன் அடிப்படையில்

சோதனையைச் செய்ய முடியும். ஆனால், எல்லா வகைகளையும் சோதனைக்கு உட்படுத்த முடியாது.

நெறிமுறை #3:

முன்கூட்டிய சோதனை(Early Testing)
வாழ்க்கையைப் போலவே தான் டெஸ்டிங்கும்! சோதனைகளை முதலிலேயே எதிர்கொள்வது சிறந்தது. ஒரு மென்பொருள் உருவாக்கத்தின் கடைசிக்கட்டத்தில் சோதனைகளைச் செய்து பிழைகளைக் கண்டுபிடிக்கலாம். ஆனால், அந்தப் பிழைகளைக் களைய ஆகும் நேரமும் உழைப்பும் அதிகமாகத் தேவைப்படும் அல்லவா? எனவே தான், கூடிய வரை சோதனையை முன் கூட்டியே தொடங்குவது என்பது சாலச் சிறந்தது. இந்த நெறிமுறையைப் பின்பற்றும் போது டெஸ்டர்களையும் முன் கூட்டியே களத்தில் இறக்கி விடலாம். டெஸ்டர்கள் முன்னதாகக் களம் இறங்க, இறங்க, மென்பொருளின் தரம் மேல் உயரும்.

நெறிமுறை #4:

பிழை கொத்து (Defect Clustering)
என்ன தலைப்பு இது? பிழைகளைக் கொத்துவது தானே டெஸ்டரின் வேலை? என்கிறீர்களா? இது கொத்துவது இல்லை! கொத்துகள் - அதாவது பிழைகளைக் குவிப்பது! ஒரு டெஸ்டர் எப்படிப் பிழைகளைக் கொத்திக் குவிப்பது என்பதைப் பற்றியது!
உங்களுக்கு இத்தாலியைச் சேர்ந்த பொருளியல் அறிஞர் பரேட்டோவைத் தெரியுமா? பரேட்டோ - (Pareto) எங்கேயோ கேள்விப்பட்ட பெயர் போல இருக்கிறது என்கிறீர்களா? கேள்விப்பட்டிருந்தால் கூடப் போதும்! அவருடைய புகழ்பெற்ற வாக்கியம் ஒன்றிருக்கிறது - 80 % அளவுச் சொத்துகள் 20% மக்களிடம் தாம் இருக்கின்றன என்றார் அவர். உண்மை தானே! என்கிறீர்களா? இது மட்டுமில்லை! அவருடைய தோட்டத்தில் விளைந்த கடலையில் 80% கடலை 20% செடிகளில் விளைந்திருந்ததாக அறிந்தார் பரேட்டோ. 1930-40 களில் அமெரிக்க மேலாண்மை வல்லுநர் முனைவர் ஜோசப் ஜூரன் (Joseph Juran) பரேட்டோவின் இந்த விதி அனைத்து விடயங்களுக்கும் பொருந்தும் என்று கூறினார். இதற்குப் பரேட்டோ கொள்கை என்னும் பெயரை அவரே சூட்டினார். இந்த விதி தான் பின்னர் 80/20 விதி என்று பலராலும் அழைக்கப்பட்டது.
இப்போது எதற்கு இந்த 80/20 விதி பற்றிய கதை - என்கிறீர்களா?
பரேட்டோவின் இந்த விதி தான் எல்லா விடயங்களுக்கும் பொருந்தும் என்று கண்டுபிடித்து விட்டார்களே! அப்படியானால், நம்முடைய மென்பொருள் சோதனைக்குப் பொருந்தாதா? பொருந்தும் தானே! 80% பிழைகளுக்குக் காரணம் நிரலர் உருவாக்கும் மென்பொருளின் 20% பகுதிகள் தாம்! அவற்றைக் கண்டுபிடித்தால் போதும் - பெரும்பாலான பிழைகளைக் கண்டுபிடித்து விடலாம். இந்த விதிக்கேற்ப பிழைகளைக் கொத்துகளாக(clusters)ப் பிரித்துக் கொள்ள வேண்டும் என்பது தான் இந்த நெறி கூறும் கருத்து.

நெறிமுறை #5: பூச்சிவிரட்டலில் புதிய முறைகள் (Pesticide Paradox)

மென்பொருள் உருவாக்கம் என்பது காலத்திற்கேற்ப மாறுகின்ற ஒன்று. ஒரு காலத்தில் இணையத்தளம் வடிவமைப்பே பெரிய விடயமாகப் பார்க்கப்பட்டது. ஆனால், இன்றோ பிஹெச்பி(PHP) முதலிய கட்டற்ற மென்பொருட்கள், வேர்டுபிரஸ், ஜூம்லா போன்ற இணையத்தள வடிவமைப்புக் கட்டுமானங்கள் ஆகியன வந்து விட்டன. இதனால் ஒரு நாள், இரண்டு நாட்களிலேயே இணையத்தளங்கள் வடிவமைக்கப்பட்டு விடுகின்றன.
முன்னர் எல்லாம் இணையத்தளத்தைச் சோதிக்க வேண்டும் என்றால், கணினிகள், மடிக்கணினிகள் ஆகியவற்றில் ஒழுங்காக இயங்குகிறதா என்று பார்த்தால் போதும்! அதற்கேற்ப டெஸ்ட் கேஸ்கள் எழுதி வைத்திருந்தால் போதும். ஆனால் இப்போது அப்படியா? ஓர் இணையத்தளம் கணினியில் ஒழுங்காகத் திறக்கிறதா? மடிக்கணினியில் ஒழுங்காகத் தெரிகிறதா? அலைபேசிகளில் ஒழுங்காகத் தெரிகிறதா? என்று எவ்வளவு சோதனைகள்! இந்தச் சோதனைகள் எல்லாவற்றிற்கும் ஏற்ப எவ்வளவு டெஸ்ட் கேஸ்கள்! இதைத் தான் பூச்சி விரட்டலில் புதிய முறைகள் என்கிறார்கள். அதாவது, பூச்சிகளுக்கு ஏற்ப, அவை பயிரைத் தாக்காமல் இருக்கும் புதிய முறைகளை வேளாண்மையில் எப்படிக் கண்டுபிடிக்க வேண்டுமோ, அதே போல, மென்பொருளிலும் புதிய பூச்சிகளை(Bugs) விரட்ட, புதிய புதிய டெஸ்ட் கேஸ்களை எழுதி வைத்துக் கொள்ள வேண்டும். பழைய டெஸ்ட் கேஸ்களை மட்டுமே வைத்து சோதித்துக் கொண்டிருக்கக் கூடாது என்பது தான் இந்த நெறிமுறை!

நெறிமுறை #6:

சூழலைப் பொருத்து சோதனையை மாற்றுக (Testing is context dependent)
மென்பொருள் சோதனையைப் பொருத்தவரை, ஒரு டெஸ்டர் - பயனரைப் போலச் சிந்திக்க வேண்டும்; செயல்பட வேண்டும். பயனர் என்பவர் துறைக்குத் துறை வேறுபடுபவர். பணப் பரிமாற்றத்திற்கான செயலியைப் பயன்படுத்துபவரையும் வாட்சப் போன்ற சமூக வகை செயலியைப் பயன்படுத்துபவரையும் ஒரே அளவுகோல் கொண்டு அளக்க முடியாது அல்லவா?
வாட்சப் செயலியைப் பயன்படுத்துபவர் ஒவ்வொரு முறை செயலியைப் பயன்படுத்தும் போதும் கடவுச்சொல் கேட்கப்படுவதை விரும்ப மாட்டார். ஆனால், பணப் பரிமாற்றத்திற்கான செயலியைப் பயன்படுத்துபவர் - ஒவ்வொரு முறை பயன்படுத்தும் போதும் கடவுச்சொல் கேட்கப்பட வேண்டும் என்று விரும்புவார் அல்லவா? ஆக, இங்கு நமக்கு இருப்பது இருவேறு வகைச் சூழல்கள்! இந்தச் சூழல் ஒவ்வொன்றையும் ஒவ்வோர் அளவுகோல் கொண்டு சோதிக்க வேண்டும் அல்லவா? இதைத் தான் சூழலுக்கு ஏற்ப சோதனையை மாற்றுக என்று நெறிப்படுத்திச் சொல்கிறார்கள்.

நெறிமுறை #7: பிழை வரா நிலை - சோதனையே பிழை! (Absence of error fallacy)

ஒரு மென்பொருளைச் சோதிக்கத் தொடங்குவதற்கு முன்பு, எந்த வகை தேவைகளைச் சோதிக்கப் போகிறோம் என்பதை நன்கு உணர்ந்து, புரிந்து கொண்டு சோதிக்கத் தொடங்க வேண்டும். வாடிக்கையாளரின் தேவைகளைத் தவறாகப் புரிந்து கொண்டால் - சில நேரங்களில் மென்பொருளில் பிழையே இல்லையே! என்று தோன்றும். அது ஒரு வகை தோற்ற மயக்கம். அதில் மயங்கிவிடக் கூடாது. பிழையே இல்லாத மென்பொருள் என்று ஒன்று உலகத்திலேயே கிடையாது. அப்படி ஒரு மென்பொருளை உருவாக்கவும் முடியாது. அதே நேரம், தவறான தேவையை மனத்தில் வைத்துக் கொண்டு பிழைகள் கண்டுபிடிப்பதும் கூடாது. நாம் நினைவில் நிறுத்த வேண்டிய உண்மைகள்

1) பிழையே வரவில்லை என்றால் சோதனை முறையை மாற்ற வேண்டும்.

2) அதே நேரம் - நாம் கண்டுபிடிக்கும் பிழைகள் - தேவை சுவட்டு ஆவணத்தின்(BRS Document) அடிப்படையில் மட்டுமே அமைய வேண்டும்.

இதுவரை நாம் பார்த்த ஏழு நெறிமுறைகள் தாம் மென்பொருள் சோதனைக்கு அடிப்படையான நெறிமுறைகள் ஆகும். இன்னும் மென்பொருளை அதன் செயல்திறன்(performance) அடிப்படையில் எப்படிச் சோதிப்பது? என்பன போன்ற கேள்விகளுக்கு அடுத்த பதிவில் விடை தேடுவோம்.

இயங்கு சோதனையும் திறன் சோதனையும்

இந்தப் பதிவில் நாம் பார்க்கப் போவது - ஒரு மென்பொருளின் இயங்குதன்மை(Functionality)யை எப்படி எல்லாம் சோதிப்பார்கள் என்பதைப் பார்ப்போம். அடுத்த பதிவில் அந்த மென்பொருளின் திறனை(Performance) எப்படிச் சோதிப்பார்கள் என்பதைப் பார்ப்போம்! முதலில் இயங்கு தன்மை என்றால் என்ன? திறன் என்றால் என்ன? அதை முதலில் சொல்லுங்கள் என்கிறீர்களா? சரி தான்! அதை முதலில் பேசி விடுவோம்.

ஒரு கதை சொல்லட்டுமா? கதிர் ஒரு மென்பொறியாளன். தமிழ்நாட்டில் பிறக்க ஓர் ஊர், பிழைக்க ஓர் ஊர். இதற்குக் கதிர் மட்டும் விதிவிலக்கா என்ன? சென்னை வந்து மூன்று மாதங்கள் ஆகிவிட்டன. திடீரென அவனுக்கு ஒரே வீட்டு நினைவு. ஒரு வாரம் ஊருக்குப் போய் வரலாம் என நினைத்து, இணையத்தின் வழி - தொடர்வண்டிப் பயணத்திற்குப் பதிய முயன்றான்.

இணையத் தளத்தில் கிளம்ப வேண்டிய இடம், சேர வேண்டிய இடம், நாள், நேரம், தொடர்வண்டி என எல்லாம் ஒழுங்காகக் கொடுத்து விட்டான். ஆனாலும் பயணச்சீட்டு உடனடியாகக் கிடைத்தபாடில்லை. ஒருவழியாகப் பயணச்சீட்டு கிடைத்த போது ஐந்து நிமிடங்கள் ஆகிவிட்டன. ‘என்ன இது இணையத்தளம்? இவ்வளவு மெதுவாக இருக்கின்றது?’ என்று சலித்துக் கொண்டான் கதிர்.

இவ்வளவு தான் கதை! என்ன – அதற்குள் கதை முடிந்து விட்டதா என்கிறீர்களா? நம்முடைய கட்டுரைக்கு இவ்வளவு கதை போதும். கதிரின் கதையைப் படித்து முடித்திருக்கும் உங்களிடம் இப்போது சில கேள்விகள்

கதிருக்கு என்ன நடந்தது?

1. கதிரால் எதிர்பார்த்த ஊருக்கு, எதிர்பார்த்த வண்டியில், எதிர்பார்த்த நேரத்தில் பயணச்சீட்டு எடுக்க முடியவில்லை.

அ. ஆம் ஆ. இல்லை.

2. கதிரால் விரும்பியபடி பயணச்சீட்டு எடுக்க முடிந்தது - ஆனால் அதற்கு ஆன நேரம் அதிகம் என அவன் நினைத்தான்.

அ. ஆம் ஆ. இல்லை

முதல் கேள்விக்கு உங்கள் பதில் என்ன? கதைப்படி, கதிருக்குப் பயணச்சீட்டு கிடைத்து விட்டது. எனவே, இதற்கான சரியான பதில் ஆ. இல்லை.

இரண்டாவது கேள்விக்கு உங்கள் பதில் என்ன? ஒரு சாதாரண பயணச்சீட்டுக்கு ஐந்து நிமிடங்கள் ஆகின்றனவே எனக் கதிர் நினைத்தான் அல்லவா? எனவே, இந்தக் கேள்விக்கான பதில் அ. ஆம்.

இந்தக் கதைக்குக் கேட்கப்பட்ட இரண்டு கேள்விகளில்

- முதல் கேள்வி இணையத்தளத்தின் இயங்குதன்மை(Functionality) பற்றியது.
- இரண்டாவது கேள்வி இணையத்தளத்தின் திறன்(Performance) பற்றியது.

இந்தக் கதையில் இருந்து நமக்குத் தெரிய வருவது - பயணச்சீட்டு பதியும் இணையத்தளத்தின் இயங்குதன்மையில் எந்தச் சிக்கலும் இல்லை; ஆனால் திறன் மேம்பட்டதாக இல்லை.

ஒரு டெஸ்டராக – ஒவ்வொரு மென்பொருளின் இயங்குதன்மை(Functionality), திறன்(Performance) ஆகிய இரண்டுமே சோதிக்கப்பட வேண்டியவை அல்லவா? இவை இரண்டையும் எப்படியெல்லாம் சோதிக்கலாம் என்று பார்த்து விடுவோமா?

1) புகை சோதனை (Smoke Testing)

எப்போது புகை வரும்? சரியாகச் சொன்னீர்கள்! நெருப்பில்லாமல் புகையாது. இங்கே நெருப்பு என்பது பெரிய சிக்கலைக் குறிக்கும் அல்லவா? அதே தான் இங்கும்! ஒரு மென்பொருளில் எங்கெங்கு இருந்தெல்லாம் நெருப்பு வர வாய்ப்பு இருக்கிறதோ - அதாவது முக்கியமான இடங்களை எல்லாம் தேர்ந்தெடுத்து - அவற்றை மட்டும் முழுமையான சோதனைக்கு உட்படுத்துவது தான் புகை சோதனை (Smoke Testing) ஆகும்.
சோதனையின் போது புகை எதுவும் வரவில்லை எனில் மென்பொருள் நிலையானதாக(stable) இருக்கிறது என்று புரிந்து கொள்ளலாம். பொது நிலையில் புகை சோதனை - பயனர் ஏற்புச் சோதனை(User Acceptance Testing)க்கு முன்பு செய்யப்படும்.

2) நலச் சோதனை (Sanity Testing)

நலச் சோதனை என்பது ஒரு மென்பொருளில் புதிய இயல்புகள்(features) சேர்க்கப்படும் போது மென்பொருளின் நலம் எந்த நிலையில் இருக்கிறது என்று பார்ப்பது ஆகும். ஏனென்றால், புதிய இயல்புகள் சேர்க்கப்படும் போது - மென்பொருளின் நலத்தில் ஏதேனும் குறைகள் வந்துவிடக்கூடாது என்பது தான் இதன் நோக்கம். எனவே தான், நலச் சோதனையை முழுச் சோதனை(Regression Testing)க்கு முன்பு செய்வார்கள்.

3) முழுச் சோதனை(Regression Testing)

முழுச் சோதனை அல்லது முழுமைச் சோதனை(Regression Testing) என்பது வாடிக்கையாளரின் எல்லாத் தேவைகளும் மென்பொருளில் வடிவமைக்கப்பட்டிருக்கின்றனவா என்று பார்ப்பதாகும். சோதனை நிலையின் இறுதியில் நடக்கும் சோதனை முழுச் சோதனை ஆகும்.

4) மறு சோதனை(Retesting)

சோதனைகளில் கண்டுபிடிக்கப்படும் பிழைகளை நிரலர்கள் திருத்துவார்கள். அவர்கள் திருத்தியது சரியா என்று பார்ப்பது தான் மறு சோதனை என்று சொல்லப்படும். அதாவது, இங்கு - மொத்த மென்பொருளையும் சோதிக்காமல், பிழையை நீக்குவதற்கு நிரலர் கொடுத்த திருத்தம் சரியா என்று மட்டும் சோதிப்பது மட்டுமே நடக்கும்.

5) சித்தம் போக்கு சிவம் போக்கு

அன்பே சிவம்! சித்தம் போகும் வழி - சிவம் போகும் வழியாக இருக்க வேண்டும் என்பதே தமிழர் நெறி! சித்தம் எந்த வழியில் போகும்? (அன்பாகிய) சிவம் போகும் வழியில்! அந்த வழியின் அடிப்படை அன்பு என்று சொல்ல முடியுமே தவிர, வரையறுத்துச் சொல்ல முடியாது அல்லவா? அதை எதற்கு இங்கே சொல்கிறீர்கள் என்கிறீர்களா? காரணம் இருக்கிறது. இப்படிச் சித்தம் போகும் வழி - மென்பொருளைச் சோதிப்பது ஒரு வகைச் சோதனையாகும்.

நேர்வழியில் போய் ஒவ்வோர் அலகாக(Unit)ச் சோதிப்பது என்பது ஒரு வகை! சித்தம் போகும் வழியில் பயணித்து பிழைகளைக் கண்டுபிடிப்பது என்பது வேறொரு வகை! இந்த வகைச் சோதனைகளுக்கு கொரில்லாச் சோதனை(Guerilla Testing) அல்லது குரங்கு போல் அங்கும் இங்கும் தாவிச் செய்யும் சோதனை என்னும் அடிப்படையில் இதற்குக் குரங்குச் சோதனை (Monkey Testing) என்றும் பெயர் உண்டு.

இவை தாம் பெரும்பாலும் இயங்குதன்மையைச் சோதிக்கும் வகைகள் ஆகும். அடுத்த பதிவில், திறன் சோதனைகள் செய்வது எப்படி என்று விரிவாகப் பார்ப்போம்.

இயங்கு சோதனையும் திறன் சோதனையும் - 2

இந்தப் பதிவில் முந்தைய பதிவில் சொன்னது போல, திறன் சோதனைகளைப் பார்ப்போமா? திறன் என்றால் என்ன என்று முந்தைய பதிவிலேயே பேசிவிட்டோம் அல்லவா? எனவே நேரடியாக, அதன் வகைகள் என்னென்ன என்று பார்க்கத் தொடங்குவோமா?

1) பயன்பாட்டுச் சோதனை (Usability Testing)

கொஞ்சம் யோசித்துப் பாருங்கள் - ஒரு மென்பொருளின் வெற்றி என்பது அந்த மென்பொருள் பயன்படுத்துவதற்கு எவ்வளவு எளிதாக இருக்கிறது என்பதையும் பொருத்தது தானே! அதனால் தானே - லினக்ஸ் போன்ற பெரிய பெரிய தலைகள் கூட, விண்டோஸ் போன்ற வில்லன்களிடம் தொடக்கத்தில் அடி வாங்கியிருந்தார்கள். எந்த ஒரு மென்பொருளையும் பயன்படுத்துவதற்கு எளிதாக இருக்கிறதா என்று பார்ப்பதும் எந்த வகை பயனருக்கு மென்பொருளை உருவாக்குகிறோமா அவர்கள் பயன்படுத்த எளிதாக இருக்கிறதா என்று பார்ப்பதும் தான் பயன்பாட்டுச் சோதனை ஆகும்.
முகநூல், கட்செவி(வாட்சப்), யூடியூப் போன்ற சமூக தளங்களின் வெற்றிக்குப் பின்னால் பயன்பாட்டு எளிமை இருப்பதை மறுக்க முடியாது அல்லவா?

2) பளு தாங்கும் சோதனை(Load Testing)

இதென்ன சோதனை என்கிறீர்களா? சின்னச் சின்ன மென்பொருட்களுக்குப் பளு தாங்கும் சோதனை தேவையில்லை தான்! ஆனால், பல பயனர்கள் ஒரே நேரத்தில் பயன்படுத்தும் இணையம் சார்ந்த எல்லா மென்பொருட்களும் அத்தனைப் பயனர்களின் உள்ளீடு, வெளியீட்டுப் பளுக்களை மென்பொருள் தாங்குகிறதா என்று சோதிக்க வேண்டும் அல்லவா? எடுத்துக்காட்டாக, ஐஆர்சிடிசி(IRCTC) போன்ற ஒரு தளத்தில் ஒரே நேரத்தில் பத்தாயிரம் பேர் முன்பதிவுக்கு வந்தார்கள் என்றால், ஐஆர்சிடிசி தளம் எப்படி இயங்கும் என்பதைச் சோதிக்க வேண்டும் அல்லவா? அதாவது, பத்தாயிரம் விண்ணப்பங்களைத் தாங்கும் பளு - மென்பொருளுக்கு இருக்கிறதா? இல்லையா? என்று பார்ப்பது! அதைத் தான் பளு தாங்கும் சோதனை(Load Testing) என்கிறார்கள்.
இப்போது உங்கள் மனத்தில் எழும் கேள்வி - அதெப்படி பத்தாயிரம் பேர் முன்பதிவு செய்வதைச் சோதிப்பது? பத்தாயிரம் கணினிகள் கொண்டா? இது எப்படி நடைமுறைச் சாத்தியம்? என்று தானே கேட்கிறீர்கள்? சரி தான் - உங்கள் கேள்வி - இது போன்ற சோதனையைப் பத்தாயிரம் பேரைக் கொண்டோ பத்தாயிரம் கணினிகளைக் கொண்டோ செய்வது நடைமுறைக்குச் சாத்தியமே இல்லாத ஒன்று தான்! இந்தப் பத்தாயிரத்தை - பத்தாயிரம் விண்ணப்பங்களாக(Request) ஒரே கணினியில் இருந்து அனுப்புவார்கள்.

இப்படிப்பட்ட சோதனைகள் செய்வதற்கென்று தனியே திறன் சோதனை(Performance Testing) மென்பொருட்கள் இருக்கின்றன. ஜேமீட்டர் போன்ற பல கட்டற்ற மென்பொருட்கள் இவ்வகைச் சோதனைக்கு உதவுகின்றன.

கூடுதல் பாரச் சோதனை(Stress Testing)

ஒவ்வொரு மென்பொருளை உருவாக்கும் போதே, அது எவ்வளவு பளு(Load) தாங்க வேண்டும் என்பதைத் தீர்மானித்தே உருவாக்குவார்கள். பொது நிலையில், திறன் சோதனை என்பது பலர் ஒரே நேரத்தில் பயன்படுத்தும் இணையம் சார்ந்த மென்பொருட்களுக்குத் தான் தேவைப்படும். அந்தச் சூழலில் திருவிழாக் காலங்கள் போன்ற நேரத்தில் பளுவைத் தாண்டி நிறைய விண்ணப்பங்கள் வர வாய்ப்பிருக்கிறது அல்லவா? அப்படிப் பளுவைத் தாண்டி விண்ணப்பங்கள் வரும் போது மென்பொருட்கள் எப்படிச் செயல்படும்? மெதுவாக இயங்குமா? இல்லை இயங்காமல் படுத்துவிடுமா? என்றெல்லாம் சோதித்துப் பார்க்க வேண்டும் அல்லவா? அதற்குத் தான் கூடுதல் பாரச் சோதனை செய்வார்கள்.

பாதுகாப்புச் சோதனைகள் (Security Testing):

அண்மைக் காலங்களில் இணையப் பயன்பாடு அதிகமாக பின்னர், பாதுகாப்புச் சோதனைகள் (Security Testing) என்பது தனிப் பிரிவாக வளர்ந்து வருகிறது. சரியான பயனர் தாம் பயன்படுத்துகிறாரா என்பதைப் பயனர் பெயர், கடவுச்சொல், ஒரே ஒரு நேரம் மட்டும் பயன்படும் கடவுச்சொல் (One Time Password), இவை எல்லாம் தாண்டி, பயனரின் தட்டச்சு வேகத்தைக் கொண்டே யார் தட்டச்சிடுகிறார்கள் என்று கண்டுபிடிப்பது என்று பல்வேறு வழிகளில் கண்டுபிடிக்கிறார்கள். இந்த வகைச் சோதனைகளுக்குச் சான்று உறுதிச் சோதனை (அதாவது, கொடுக்கும் சான்றுகள் உறுதியானவை தாமா? என்று பார்ப்பது - Authentication Testing) என்று பெயர்.

அதிகார உறுதிச் சோதனை (Authorization Testing):

சான்று உறுதிச் சோதனையைப் போலவே அதிகார உறுதிச் சோதனை (Authorization Testing) என்று ஒன்று இருக்கிறது. அமேசான் முதலிய இணையத்தளங்களை எடுத்துக் கொள்ளுங்கள். அந்தத் தளத்திற்குப் போய் பொருள் வாங்குபவர்க்கும் தனிக்கணக்கு வைத்திருப்பார்கள். பொருள் விற்பவர்களுக்கும் தனிக் கணக்கு வைத்திருப்பார்கள். ஆனால் இருவருக்கும் வெவ்வேறு உரிமைகள் கொடுக்கப்பட்டிருக்கும் அல்லவா? பொருள் விற்பவர் - பொருள்களைத் தளத்தில் சேர்க்கும் உரிமை கொண்டிருப்பார். ஆனால் வாங்குபவர்க்கு அந்த உரிமை இருக்காது. இப்படி, இருவருக்குமே கணக்குகள் இருந்தாலும் யார் யாருக்கு என்னென்ன அதிகாரங்கள் கொடுக்கப்பட்டிருக்கின்றன என்று பார்ப்பதும் ஒவ்வொரு வகை பயனரையும் அதற்கேற்ப சோதிப்பதும் தான் தான் அதிகார உறுதிச் சோதனை(Authorization Testing) என்று சொல்லப்படுகிறது.

உலகமயமாக்கல் சோதனை (Globalization Testing)

உலகமயமாக்கலா? மென்பொருள் சோதனையிலுமா? என்று கேட்கிறீர்களா? பொருளியல் உலகமயமாக்கல் என்பது வேறு மென்பொருள் உலகமயமாக்கல் என்பது வேறு! இங்கு உலகமயமாக்கல் என்று சொல்வது - எந்த ஊரில் இருந்து நீங்கள் ஓர் இணையத்தளத்தைப் பார்க்கிறீர்களோ - அந்த ஊருக்கு ஏற்ற பொருத்தமான தளத்தைக் காண்பிப்பது! அதாவது, நீங்கள் google.com என்று தட்டச்சிட்டாலும் தானாகவே இந்தியாவுக்குரிய google.co.in காண்பிக்கப்படுவது, உங்கள் உலாவியின்(Browser) மொழியைப் பொருத்து இணையத்தளங்களின் மொழியும் தானாகவே மாறுவது ஆகியவற்றைச் சோதிப்பது தான் உலகமயமாக்கல் சோதனை என்று சொல்வார்கள். இதை எப்படி உலகமயமாக்கல் என்று சொல்ல முடியும்? இது உள்ளூர்மயமாக்கல்(Localization) ஆக அல்லவா இருக்கிறது என்று கிடுக்கிப் பிடியாகக் கேட்பவர்களும் இருக்கிறார்கள். அவர்கள் கேட்பதும் சரிதானே! எனவே, இவ்வகைச் சோதனையை உள்ளூர்மயமாக்கல்(Localization Testing) என்றும் சொல்லலாம்.

இன்னும் மென்பொருள் சோதனை பற்றித் தெரிந்து கொள்ள வேண்டிய முதன்மையான பகுதி ஒன்று இருக்கிறது - அது தான் மென்பொருள் வாழ்க்கை வட்டமும் அதை நடைமுறைப்படுத்தும் முறைகளும்! அவற்றை வரும் பகுதிகளில் பார்ப்போம்.

மென்பொருள் வாழ்க்கை வட்டமும் நடைமுறைகளும்

மென்பொருள் வாழ்க்கை வட்டம்(Software Development Life Cycle) என்பது வாடிக்கையாளரிடம் மென்பொருளுக்கான திட்டத்தை வாங்குவதில் தொடங்கி, நிறைவாக, மென்பொருளை ஒப்படைப்பதில் முடிகிறது. இதில் மென்பொருள் வாழ்க்கை வட்டத்தை எங்கே தொடங்குவது என்பதை ஏற்கெனவே பார்த்திருக்கிறோம். தொடக்கத்தில் உருவாக்கப்படும் ஆவணம் என்ன என்பதையும் நாம் ஏற்கெனவே பார்த்து விட்டோம். அந்த ஆவணத்தில் அடிப்படையில் முறையாகத் திட்டமிட வேண்டும். திட்டமிடலின் அடிப்படையில் நிரலர்கள்(Developers) மென்பொருளை வடிவமைப்பது (Design), உருவாக்குவது(Develop) ஆகிய வேலைகளில் ஈடுபடத் தொடங்குவார்கள். வடிவமைப்பு, உருவாக்கம் ஆகியன ஒரே ஒரு நிரலர் செய்யும் வேலை இல்லை; பலரது கூட்டு முயற்சி! ஓர் அணியாகத் தான் செய்ய முடியும். ஒவ்வொரு நிரலரும் தாம் உருவாக்கிய அலகை(Module/Unit) முறையாகச் சோதித்து கிட் போன்ற தளத்தில் ஏற்றி விடுவார்கள். பிறகு ஒருங்கிணைப்புச் சோதனை(Integration Testing) நடத்தப்படும். இதைப் பற்றி விரிவாக நாம் ஏற்கெனவே படித்திருக்கிறோம்.

ஒருங்கிணைப்புச் சோதனைக்குப் பிறகு, முழுமையான சோதனை(Regression Testing) நடத்தப்படும். அதன் பிறகு பயனர் ஏற்புச் சோதனை(User Acceptance Testing) நடத்தப்பட்டு மென்பொருள் வெளியிடப்படும். சுருக்கமாகச் சொல்ல வேண்டும் என்றால்,

1) வாடிக்கையாளர் தேவைகளைப் பெறுதல் (Requirements Gathering)
2) தேவைகளை ஆராய்தல் (Requirements Analysis)
3) திட்டமிடல்(Planning)
4) வடிவமைப்பு, உருவாக்கம் (Design and Development)
5) சோதனையிடல் (Testing)
6) வெளியிடல், பேணல் (Implementation and Maintenance)

ஆகியன தான் மென்பொருள் வாழ்க்கை வட்டத்தின் படிநிலைகள் ஆகும்.

இந்தப் படிநிலைகளை எப்படிச் செயல்படுத்துவது என்று சொல்பவை தாம் சீர்வடிவங்கள்(Models) ஆகும். பல்வேறு மென்பொறியாளர்களின் பட்டறிவின் அடிப்படையில் பல வகை சீர்வடிவங்கள் நடைமுறையில் இருக்கின்றன. அவற்றுள் தலையாய சீர்வடிவங்கள்

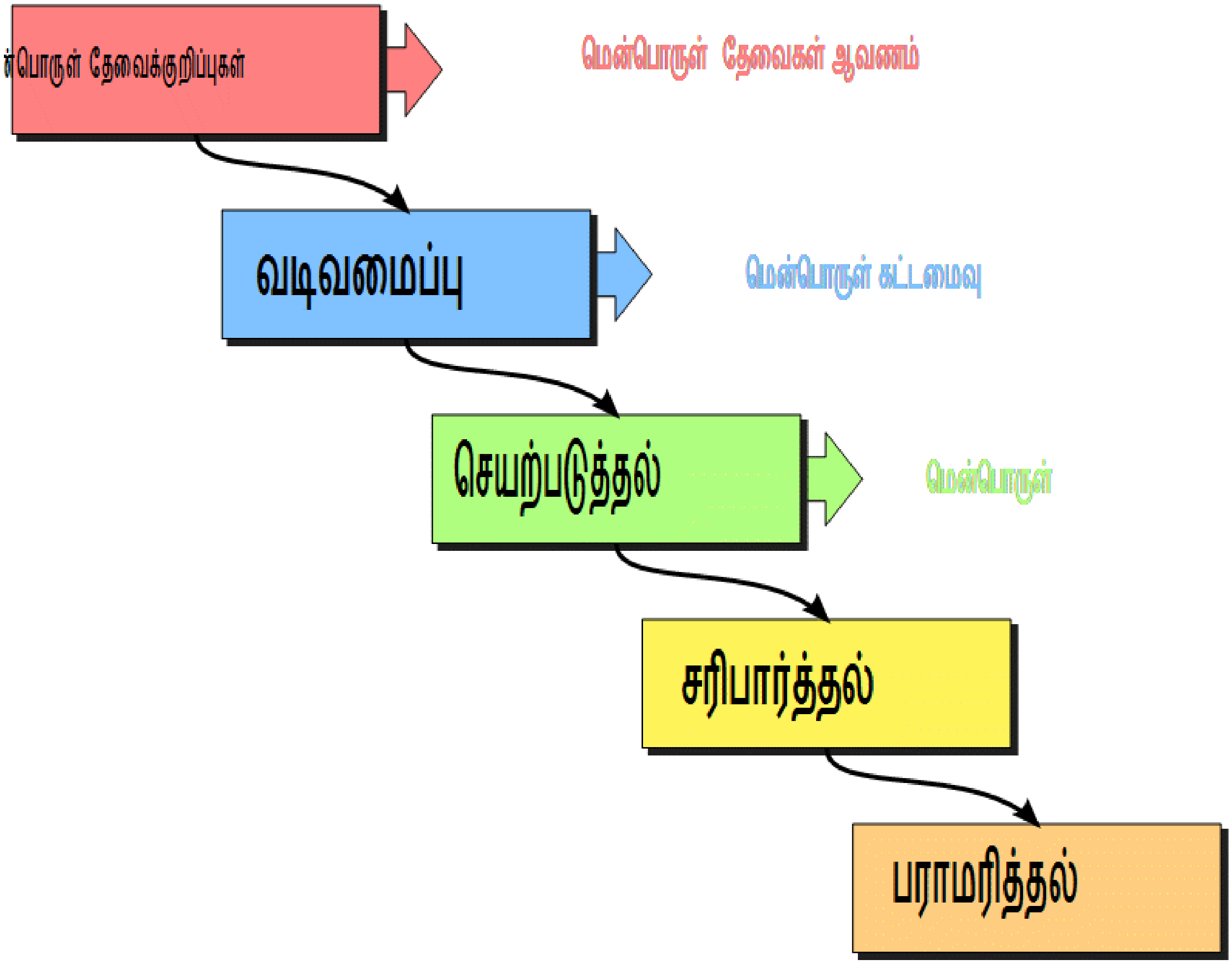
1) அருவி(Waterfall) முறை
2) ‘வி' (V) முறை
3) தகவெளிமை(Agile) முறை ஆகியனவாகும்.

1) அருவி(Waterfall) முறை

அருவியில் நீர் எப்படிப் பாயும்? மேல் இருந்து கீழாகத் தானே! அதே போன்ற முறை தான் இந்த முறை! மென்பொருள் வாழ்க்கை வட்டத்தின் ஒரு படியில் இருந்து அடுத்த படிக்கு இறங்கி விட்டால் பிறகு, மேலே போகக் கூடாது. அதாவது, வாடிக்கையாளரிடம் எல்லாத்

தேவைகளையும் கேட்டுத் தொகுத்த பின்னர், அடுத்து தேவைகளை ஆராயலாமே தவிர, திரும்பவும் வாடிக்கையாளரிடம் போய்த் தேவைகள் பற்றிப் பேசக் கூடாது. இதே போல் தான் அடுத்தடுத்த படிகளும்! மென்பொருளை உருவாக்கிச் சோதனையிடல் படிக்குப் போன பிறகு, திரும்பவும் உருவாக்குதல் படிக்கு ஏறக்கூடாது. இது தான் அருவி(Waterfall) முறை ஆகும்.

இந்த முறையை அறிமுகப்படுத்தியவர் அமெரிக்கக் கணினியாளர் வின்ஸ்டன் டபிள்யூ. ராய்ஸ் ஆவார். (பட உதவி: [விக்கிப்பீடியா])



எங்குப் பயன்படுத்தலாம்?

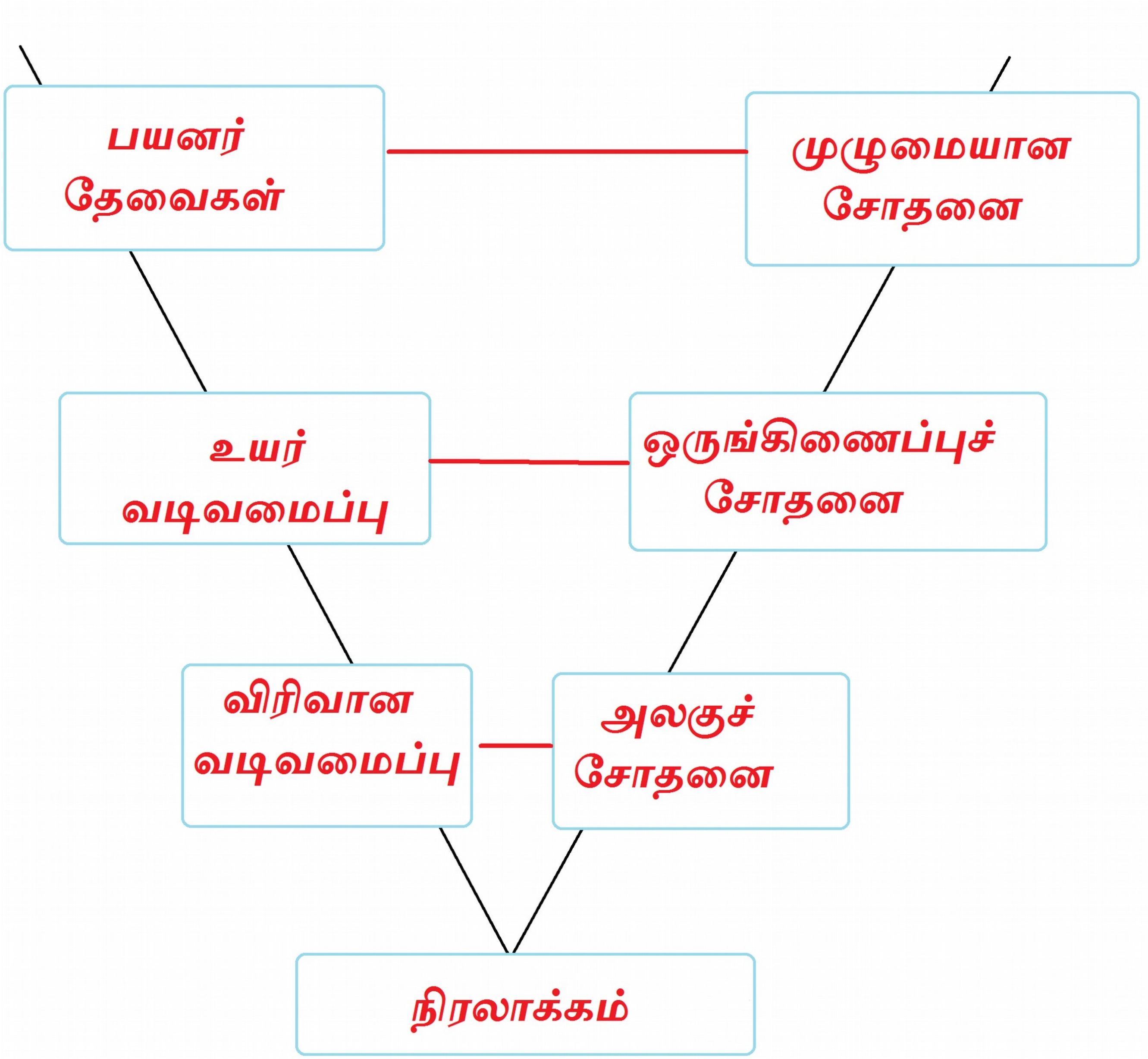
அருவி முறையில் அதிகமாகக் கவனிக்க வேண்டிய ஒன்று - ஒரு படியில் இருந்து இறங்கிவிட்டால், மீண்டும் மேலே ஏற முடியாது என்பதைத் தான்! எனவே இந்த நடைமுறை எங்கு பொருந்துகிறதோ அங்கு அருவி முறையில் மென்பொருள் வாழ்க்கை வட்டத்தை அமைக்கலாம். எடுத்துக்காட்டாக, ஏற்கெனவே உருவாக்கப்பட்டு பயன்பாட்டில் உள்ள ஒரு மென்பொருளில் சின்ன மாற்றம் ஒன்றை ஏற்படுத்த வேண்டும் என்றால் அருவி முறையைப் பயன்படுத்தலாம். எ.காட்டாக, விக்கிப்பீடியா - இந்திய மொழிகளில் புதிய மொழி ஒன்றில் தன்னுடைய பக்கங்களை விரிவாக்குகிறது என்று

வைத்துக்கொள்ளுங்கள். ஏற்கெனவே தமிழ் முதலிய மொழிகளில் விக்கிப்பீடியா இருக்கிறது அல்லவா? அதாவது, விக்கிப்பீடியா ஏற்கெனவே நிலையான நிரல்களால் உருவாக்கப்பட்டிருக்கிறது. இப்போது நடப்பது விரிவாக்கம் மட்டுமே! இது போன்ற சூழலில் அருவி முறையைச் செயல்படுத்தலாம்.

எங்கு உதவாது?

அருவி முறையைச் செயல்படுத்த வேண்டும் எனில், தொடக்கம் முதல் ஒவ்வொரு படியும் முறையாக, முழுமையாக முடிக்கப்பட வேண்டும் - அதாவது, வாடிக்கையாளர் தேவைகளை நிரலர் கேட்பதில் தொடங்கி, மென்பொருள் வெளியீடு வரை! ஆனால், பெரும்பாலான நேரங்களில் வாடிக்கையாளருக்குத் தம்முடைய தேவைகளை முழுமையாகப் பட்டியலிட முடியாது. மென்பொருளை உருவாக்க, உருவாக்க, வாடிக்கையாளர் - தம்முடைய தேவைகளைக் கூட்டுவது தான் பெரும்பாலான நடைமுறை. எனவே, அருவி முறை பெரும்பாலும் புதிய திட்டப்பணிகளுக்கு(Project)ப் பொருந்தாது. இது தவிர்த்து, இந்த முறையில் டெஸ்டர்களின் பங்களிப்பு கிட்டத்தட்ட கடைசிக்கட்டத்தில் தான் தொடங்குகிறது. டெஸ்டர்கள் எவ்வளவு முன்னர் களம் இறங்குவார்களோ தரத்தின் அளவு அவ்வளவு நன்றாக இருக்கும். அருவி முறையில் டெஸ்டர்கள் கடைசியாகக் களம் இறங்குவது ஒருவகையில் சின்ன இழப்புத் தான். அருவி முறையில் உள்ள இந்தச் சிக்கலைப் போக்க வந்த சீர்வடிவம் தான் 'வி'(V) சீர்வடிவம்!

'வி'(V) சீர்வடிவம்:



முன்னர் பார்த்த அருவி முறையைப் போன்றதே தான் வி முறையும்! ஆனால் ஒரு குறிப்பிடத்தகுந்த வேறுபாட்டுடன் - டெஸ்டர்கள் முன்னரே களம் இறக்கிவிடப்படுவார்கள் என்பதே அது! அதைத் தான் மேல் உள்ள படம் விளக்குகிறது.

1) பயனர் தேவைகளை வாங்கி, நிரலர்கள் ஆராயும் அதே நேரத்தில் - டெஸ்டர்கள் முழுமைச் சோதனை(Regression Testing)க்கான டெஸ்ட் கேஸ்களை எழுதுவார்கள்.

2) தேவைகளின் அடிப்படையில் உயர் வடிவமைப்பில் (High Level Design) நிரலர்கள் ஈடுபடும் போது, அந்த உயர் வடிவமைப்புக்கேற்ற டெஸ்ட் கேஸ்களை டெஸ்டர்கள் எழுதுவார்கள்.

3) பின்னர் நிரலர்கள் - ஒவ்வோர் அலகுக்கும் விரிவான வடிவமைப்பை(Low Level Design) உருவாக்கும் போது - டெஸ்டர்கள் ஒருங்கிணைப்புச் சோதனைக்கான (Integration Testing) டெஸ்ட் கேஸ்களை எழுதி வைப்பார்கள்.
4) நிரலர்கள் நிரல் எழுதி முடிக்கும் போது, டெஸ்டர்கள் அந்தந்த வகை டெஸ்ட் கேஸ்களை எடுத்து சோதனைகளைச் செய்யத் தொடங்குவார்கள்.
இந்த முறையில், வடிவமைப்பு(Design) நடந்து கொண்டிருக்கும் போதே, டெஸ்டர்கள் டெஸ்ட் கேஸ்களை எழுதத் தொடங்குகிறார்கள் அல்லவா? அருவி முறையைப் போல் அல்லாது, இப்படித் தான் இங்கு டெஸ்டர்கள் முன்னரே களம் இறக்கிவிடப்படுகிறார்கள். இப்படித் தான் வி முறையில் தர மேம்பாடு கொடுக்கப்படுகிறது.
‘வி' என்றால் என்ன?
‘வி' என்று ஆங்கிலத்தில் சொல்லப்படும் இந்த முறையின் ஒரு இடப்பக்கம் உள்ள வேலைகள் நடக்கும் போது டெஸ்டர்கள் டெஸ்ட் கேஸ்கள் எழுதுவார்கள் அல்லவா? அந்த நிலையைச் 'சரிபார்த்தல்' (Verification) என்று சொல்வார்கள். பின்னர் நிரலாக்கம் முடிந்த பிறகு, டெஸ்டர்கள் சோதிப்பார்கள் அல்லவா? அந்த நிலைக்குச் 'செல்லத்தக்கதாக்குதல்' (Validation) என்று பெயர். Verification, Validation ஆகிய இரண்டிலும் உள்ள முதல் எழுத்துகளைச் சேர்த்துத் தான் 'வி' முறை என்று அழைக்கிறார்கள்.
எங்கு பயன்படுத்தலாம்?
எங்கெல்லாம் அருவி முறையைச் செயல்படுத்த முடியுமோ அங்கெல்லாம் அருவிமுறைக்கு மாற்றாக 'வி' முறையைப் பயன்படுத்தலாம்.
என்ன குறைபாடு?
‘வி' முறையும் அருவி முறையில் உள்ள பிற குறைகளைத் தன்னகத்தே கொண்டிருக்கிறது. அதாவது, ஒவ்வொரு படியும் முடிந்த பிறகு தான் அடுத்த படிக்குத் தாவ முடியும் என்னும் குறைபாடு வி முறையிலும் இருக்கிறது தானே!
இன்னும் அண்மைக் காலங்களில் எல்லா இடங்களிலும் பயன்படுத்தப்படும் தகவெளிமை(Agile) முறை பற்றிப் பார்க்க வேண்டியிருக்கிறது. அடுத்த பதிவில் அதைப் பார்ப்போமா?

தகவெளிமை முறை (Agile Methodology)

அண்மைக்காலங்களில் பல மென்பொருள் நிறுவனங்கள் தகவெளிமை முறைக்கு மாறியிருக்கின்றன. ஏன் இந்த மாற்றம்? அப்படி என்ன இருக்கிறது இந்த முறையில்? இதைப் பற்றி விரிவாகத் தெரிந்து கொள்ள வேண்டுமே என்று நினைப்பவர்கள் கணியத்தில் திரு. அசோகன் அவர்கள் எழுதியுள்ள "எளிய தமிழில் Agile/Scrum” மின் நூலில் படித்துத் தெரிந்து கொள்ளலாம். விரிவாகத் தெரிவதற்கு முன்னர், தகவெளிமை(Agile) பற்றிய ஓர் அறிமுகமாவது வேண்டாமா? என்று கேட்பவரா நீங்கள்? அப்படியானால் உங்களுக்கானது தான் இந்தப் பதிவு!

இதற்கு முன்பு நாம் பார்த்த சீர்வடிவங்கள் இரண்டும் நீண்ட காலத் திட்டப்பணிகளுக்கு ஏற்றவை. நீண்ட காலம் என்று இங்கு சொல்லப்படுவது மாதக்கணக்கிலோ ஓரிரண்டு ஆண்டுகளிலோ செய்து முடிக்கப்படும் திட்டப்பணிகள். ஆனால் கொஞ்சம் யோசித்துப் பாருங்கள் - இன்றைய காலத்தில் அலைபேசி செயலி ஒன்றைச் செய்து முடிப்பதற்கோ ஓர் இணையத்தளத்தை வடிவமைப்பதற்கோ ஆண்டுக்கணக்கில் நேரம் எடுக்க மாட்டார்கள் அல்லவா? ஓரிரு வாரங்கள், கூடிப்போனால் ஒரு மாதத்தில் முடித்து விடுவார்கள். 2,3 வாரங்களில் உருவாக்கப் போகும் ஒரு மென்பொருளுக்குப் பல நாட்கள் வாடிக்கையாளரிடம் பேசித் தேவைகளைப் பக்கம் பக்கமாகக் கட்டுரை எழுத வேண்டிய தேவை இருக்காது தானே!

இதையெல்லாம் செய்வது தான் தகவெளிமை(Agile) முறை! தகவெளிமை முறையில் ஆவணமாக்கல் என்று ஒன்று பெரிதாகக் கிடையாது. தேவை ஏற்படும் போதெல்லாம் வாடிக்கையாளரிடம் பேசுவார்கள். அவர் சொல்வதைக் குறித்து வைத்துக் கொள்வார்கள். அந்தக் குறிப்புகளுக்குத் தான் "காப்பியம்"(epic) என்று பெயர். காப்பியம் என்றால் தான் நமக்குத் தெரியுமே! இராமாயணம், மகாபாரதம் ஆகியன காப்பியங்கள்! காப்பியங்கள் என்றாலே அதில் இருக்கும் ஒவ்வொரு கதை மாந்தருக்கும் ஓர் உள் கதை பிரிந்து போகும் அல்லவா! அதே தான் இங்கேயும்!

ஒவ்வொரு காப்பியத்தையும் நிரலர் அணி உட்கார்ந்து ஆராயும். ஆராய்ந்து பயனர் என்னென்னவெல்லாம் கேட்டிருக்கிறார் என்று ஒரு பட்டியலை உருவாக்கும். பயனர்க்குத் தேவைப்படும் அந்தக் கதைகளுக்குப் பயனர் கதைகள்(User Stories) என்று பெயர். அதாவது, ஒவ்வொரு காப்பியமும் பயனரின் பல கதைகளை(தேவைகளை)க் கொண்டிருக்கும்.

பல கதைகளா? அப்படியானால் ஒவ்வொரு கதைக்கும் நிறைய வேலைகள் செய்ய வேண்டியிருக்குமே! எந்த அடிப்படையில் - எந்த வேலையை எப்போது முடிப்பது என்று முடிவு செய்வார்கள் என்கிறீர்களா? கவலையை விடுங்கள். அதை வாடிக்கையாளரிடமே கேட்டு விடலாமே!

“நாங்கள் இந்தத் திட்டப்பணியை ஆறு வாரத்தில் முடிக்கத் திட்டமிட்டிருக்கிறோம். முதல் இரண்டு வாரங்களில் மென்பொருளின் குறிப்பிட்ட சில பகுதிகளை முடித்து நீங்கள் அப்படியே பயன்படுத்தும் வகையில் (Shippable Product) கொடுத்து விடுகிறோம். எந்தெந்தப் பகுதிகளை முடிக்க வேண்டும் என்று சொல்லுங்கள்.” என்று வாடிக்கையாளரிடமே கேட்டு விடுவார்கள். வாடிக்கையாளரை "மென்பொருள் உரிமையாளர்" (Product Owner) என்று சொல்வார்கள்.

இப்படி மென்பொருள் உருவாக்கத்தை இரண்டு இரண்டு வாரங்களாகப் பிரித்து, ஒவ்வோர் இரு வாரத்திலும் பயன்படும் வகையிலான ஒரு மென்பொருளை(Shippable product) வாடிக்கையாளருக்குக் கொடுத்து விடுவார்கள். சில நிறுவனங்கள் இரண்டு வாரங்களாகப் பிரிப்பதற்குப் பதிலாக, ஒரு வாரம், மூன்று வாரம், 4 வாரம் என அவர்கள் வசதிக்கேற்ப பிரித்துக் கொள்வார்கள். மென்பொருள் உருவாக்க நேரத்தில் இந்தப் பிரிப்புக்குச் சிற்றோட்டம்(Sprint) அல்லது குறுவோட்டம் என்று பெயர். (பொதுவாக, வேகமாகப் பாய்ந்து செல்லும் கார் பந்தயங்களில் ஒரு சுற்று முடிப்பதை sprint என்று சொல்வார்கள்.) ஒவ்வொரு சிற்றோட்டத்திலும் ஒரு காப்பியத்தின் சில பல கதைகளை ஓடி முடித்திருக்க வேண்டும்.

எல்லாம் சரி! இந்தக் கதைகளை முடிக்க எவ்வளவு நேரம் ஆகும் என்று யார் முடிவு செய்வார்கள்? முந்தைய அருவி முறை, வி முறை ஆகியவற்றில் திட்டமிடல் என்று தனியே ஒரு படி இருந்தது. இங்கு நீங்கள் அதைப் பற்றிப் பேசவே இல்லையே! என்று கேட்டால் அது நியாயமான கேள்வி தானே!

முந்தைய முறைகளைக் காட்டிலும் தகவெளிமை(Agile) முறை முற்றிலும் மாறுவது இந்த இடத்தில் தான்! முந்தைய முறைகளில் ஒவ்வொரு நிரலரும் எவ்வளவு நேரத்தில் முடிக்க வேண்டும் என்பதை அணித்தலைவர் முடிவெடுப்பார். ஆனால், தகவெளிமை(Agile) முறையில் அப்படி இல்லை! ஒவ்வொருவரும் எவ்வளவு நேரத்தில் முடிப்பார்கள் என்பதை அவரவரே தீர்மானிப்பார்கள். ஆ! இதென்ன புதுமை! அப்படியானால், ஒரு வேலையை முடிக்க எவ்வளவு நேரம் வேண்டுமானாலும் எடுத்துக் கொள்ளலாமா - என்றால் இல்லை.

ஒருங்கிணைப்பாளர் (Scrum Master):

தகவெளிமை(Agile) முறையில் ஏழில் இருந்து ஒன்பது பேர் வரை கொண்ட அணிகளைப் பிரிப்பார்கள். இந்த அணிக்கு மொய்திரள்(Scrum) அணி(team) என்று பெயர். ஒவ்வோர் அணியிலும் ஓர் ஒருங்கிணைப்பாளர் (Scrum Master) இருப்பார். தகவெளிமை(Agile)யில் உள்ள இன்னொரு சிறப்பு இந்த ஒருங்கிணைப்பாளர் (Scrum Master) தாம்! வயதில் மூத்தவரோ, வேலையில் மூத்தவரோ தான் ஒருங்கிணைப்பாளர் (Scrum Master) ஆக இருக்க வேண்டும் என்று எந்தக் கட்டாயமும் கிடையாது. ஒருங்கிணைப்பதில் ஆர்வம் உள்ள யார் வேண்டுமானாலும் ஒருங்கிணைப்பாளராக (Scrum Master) ஆகலாம். சில திட்டப்பணிகளில் அணியில் உள்ள எல்லோருக்கும் ஒருங்கிணைப்பாளர் (Scrum Master) வேலை தெரிய வேண்டும் என்பதற்காக - ஒவ்வொரு சிற்றோட்டத்திற்கும்(Sprint) ஒருங்கிணைப்பாளர் (Scrum Master) மாறுவது போல் அமைத்துக் கொள்வார்கள்.

தேவைப்பட்டால் இப்படிக் கூட அமைத்துக் கொள்ளலாம். இந்த ஒருங்கிணைப்பாளரின்(Scrum Master) வேலைகள் தாம் என்னென்ன?

1) மொத்தத் திட்டப்பணிக்கான திட்டமிடல் கூட்டங்கள் நடத்துவது (Project Planning Meeting)

2) ஒவ்வொரு சிற்றோட்டத்திற்கும்(Sprint) திட்டமிடல் கூட்டங்கள் நடத்துவது (Sprint Planning Meeting)

இந்தக் கூட்டங்களில் தாம், காப்பியங்களில்(Epic) உள்ள கதைகள்(User Stories) பற்றிப் பேசுவார்கள். ஒவ்வொரு கதையையும் முடிக்க எவ்வளவு நேரம் ஆகும் என்பதை எல்லா நிரலர்களிடமும் கேட்பார்கள். இப்படிக் கேட்கும் போது ஒவ்வொரு நிரலர் கையிலும்

அ. 1,2,3,5,8,13,21 என்று பைபோனாச்சி எண்கள் அச்சிடப்பட்ட அட்டைகள்

ஆ. மிகச் சிறியது, சிறியது, நடுத்தரம், பெரியது, மிகப்பெரியது என்பன போன்ற (சட்டைக்கான அளவுகோல் போல) அச்சிடப்பட்ட அட்டைகள்

போன்றவற்றைக் கொடுத்திருப்பார்கள். ஒவ்வொரு கதையையும் பேசி முடித்த பின்னர், அந்தக் கதையை முடிக்க எவ்வளவு உழைப்புத் தேவை என்பதற்கேற்ப ஒவ்வொருவரும் தங்கள் கைகளில் உள்ள அட்டைகளைக் காண்பிக்க வேண்டும். ஒருங்கிணைப்பாளர் ஒவ்வொருவரிடமும் "நீங்கள் ஏன் இந்த எண்ணைச் சொல்கிறீர்கள்? விளக்குங்கள்" என்று காரணம் கேட்பார். அவர்கள் சொல்லும் காரணத்தின் மேல் கலந்துரையாடல் நடக்கும். கலந்துரையாடலின் இறுதியில் மொய்திரள்(Scrum) அணி(Team) அனைவரும் (மிகச் சில நேரங்களில் பெரும்பாலானோர்) ஒத்துக்கொள்ளும் முடிவை ஏற்றுக்கொள்ளும். இப்படித்தான் திட்டமிடல் இங்கு நடக்கும். இது தகவெளிமை(Agile)யின் சிறப்புகளுள் ஒன்றாகும். நம்முடைய வேலைக்கான திட்டமிடலை நாமே உருவாக்குவது!

3) இந்தத் திட்டமிடல் வழி அணி இயங்குகிறதா என்பதை யார் கண்காணிப்பது? வேறொருவர், நிரலர்களைக் கூப்பிட்டு வேலையைக் கொடுத்தால் தானே கண்காணிக்க வேண்டும்? இங்கு திட்டமிடுவதும் நிரலர்கள் தாம்! யார் யார் எந்தெந்த வேலையைச் செய்வது என்று பிரித்துக் கொள்வதும் அவர்கள் தாம்! எனவே, இங்கு பகிரப்படுவது வேலை இல்லை, பொறுப்பு! அவரவர் தத்தமக்கு விரும்பிய வேலைகளை எடுத்துக் கொள்கிறார்கள். கடினமான வேலையை ஒருவர் எடுத்தால், பிற உறுப்பினர்கள் தத்தம் பொறுப்புகளை முடித்த பிறகு அவருக்கு உதவுவார்கள். அப்படியானால், யார் கடினமான பொறுப்புகளுடன் இருக்கிறார்கள், யார் எளிமையாகப் பொறுப்புகளை முடித்து விட்டார்கள் என்று மொத்த மொய்திரள்(Scrum) அணிக்கும் தெரிய வேண்டும் அல்லவா? இதற்காக ஒவ்வொரு நாளும் 10 அல்லது 15 மணித்துளிகள் அணிக்கூட்டம் போடுவார்கள். இந்தக் கூட்டத்திற்கு தினசரி கூட்டம்(Daily Standup meeting) என்று பெயர்.

இந்தக் கூட்டத்தில் மூன்றே மூன்று கேள்விகளை மட்டும் தான் எடுத்துக் கொள்வார்கள்.

க. நேற்று நான் என்ன செய்தேன்?

ங. இன்று என்னுடைய திட்டம் என்ன?

ச. என்னுடைய வேலையில் ஏதேனும் தடங்கல் இருக்கிறதா?

இந்த மூன்று கேள்விகளைத் தவிர, வேறு கேள்விகளைப் பேசக் கூடாது.

4. ஒவ்வொரு சிற்றோட்டத்தின்(Sprint) இறுதியிலும் பயன்படு மென்பொருளை(Shippable Product)ப் பயனரிடம் காண்பிக்க வேண்டும் அல்லவா? அதற்கான முன்னோட்டக் கூட்டத்தை(Demo Meeting)க் கூட்டுவது!
5. பலர் சேர்ந்து பணியாற்றும் இடம் என்றால் அரசல் புரசல்களுக்கு இடம் இல்லாமலா? அரசல் புரசல்களுடன் பாராட்டுகளுக்கும் இடம் இருக்கும் அல்லவா? சிக்கல்களைக் கூடிய வரை சீக்கிரம் தீர்க்க வேண்டும் என்று (Early Testing) என்று மென்பொருள் சோதனை நெறிமுறை சொல்கிறது. அதை மென்பொறியாளர்களே தம்முடைய வாழ்க்கையில் செயல்படுத்தாவிட்டால் எப்படி? - செயல்படுத்த வேண்டும் அல்லவா? அதற்காகத் தான், இப்படிப்பட்ட அரசல், புரசல்கள் என்னென்ன, பாராட்டுகள் என்னென்ன என்பதை அணியோடு உட்கார்ந்து பேச, ஒவ்வொரு சிற்றோட்டத்தின் முடிவிலும் குறைநிறை கூட்டம்(Retrospective meeting) நடத்தப்படும்.
தகவெளிமை(Agile) முறையில் இப்படிப் பல கூட்டங்கள் இருப்பதால், நாள் திட்டமிடலையே எட்டு மணிநேரத்திற்குப் பதிலாக, ஆறு மணிநேரத்திற்குத் தான் வைத்திருப்பார்கள். ஏனென்றால், தினசரி கூட்டத்தைத் தவிர மீதி எல்லாக் கூட்டங்களுமே ஒரு மணி நேரமோ இரண்டு மணி நேரமோ நடக்கும்.
எங்கு பயன்படும்?
குறுகிய காலத் திட்டப்பணிகளுக்குப் பயன்படும். அவ்வப்போது வாடிக்கையாளருக்குப் பயன்படு மென்பொருளை(Shippable Product) அனுப்புவதால் வாடிக்கையாளரின் எண்ணவோட்டங்கள், கருத்துகளை உடனுக்குடன் அறிந்துகொள்ளலாம். அதற்கேற்ப மாற்றங்களை அடுத்த குறுவோட்டத்திலேயே(Sprint) செய்தும் கொள்ளலாம்.
சிக்கல்கள்:
தகவெளிமையின்(Agile) மிகப் பெரிய சிக்கல் - முறையான ஆவணங்கள் இல்லாதது. இன்னொரு மிகப்பெரிய சிக்கல் - குறுகிய காலத் திட்டப்பணிகள் என்பதால் பெரும்பாலான இடங்களில் டெஸ்டருக்குப் பதிலாக, நிரலரையே(Developer) மாற்றி மாற்றிச் சோதிக்கச் சொல்கிறார்கள். அறிவியல் அடிப்படையில் ஒரு நிரலரே டெஸ்டராக மாறுவது எந்த அளவு சிறந்தது என்பது இன்னும் கணினி வல்லுநர்கள் நடுவே பேசி முடிக்கப்படாத கருத்தாகவே இருக்கிறது.

சாப்ட்வேர் டெஸ்டிங் - நிறைவாகச் சில வரிகள்

இதுவரை நாம் பார்த்திருப்பது சாப்ட்வேர் டெஸ்டிங் என்று சொல்லப்படும் மென்பொருள் சோதனையின் அடிப்படைகள். புதுநிலை மென்பொறியாளர் ஒருவருக்கு இந்த அடிப்படைகளே போதுமானவை தாம். நாம் பேசிய இந்தத் தலைப்புகள் தாம் ISTQB என்று அழைக்கப்படும் மென்பொருள் சோதனைக்கான பன்னாட்டுத் தரச் சான்றிதழ் அமைப்பின் பாடத்திட்டத்தில் இருப்பதை நீங்கள் பார்க்கலாம். நாம் பார்க்காது விடுபட்ட பகுதிகள் ஒரு சில இருந்தாலும் அவற்றை மிக எளிதாக உங்களால் புரிந்து கொள்ள முடியும். நாம் பார்த்தவை தவிர்த்து, ஒரு டெஸ்டருக்கு அடிப்படையில் தரவமைப்பு(Database) பற்றிய புரிதலும் அதன் அடிப்படை கட்டளைகளும் இன்றியமையாதவை. அவற்றைத் திருமதி. து. நித்யா அவர்கள் எழுதிய எளிய தமிழில் MySQL புத்தகம் வழி நீங்கள் படித்துக் கொள்ள முடியும். தகவெளிமை(Agile) பற்றி ஏற்கெனவே சொன்னது போலத் திரு. இரா. அசோகன் அவர்கள் எழுதிய "எளிய தமிழில் Agile/Scrum” புத்தகம் மூலம் விரிவாகத் தெரிந்து கொள்வது நல்லது. இன்னும் திருமிகு. கலாராணி அவர்கள் எழுதும் சோதனை வழி நிரலாக்கம் பற்றிய தொடர் - கணியத்திலேயே காணக் கிடைக்கிறது. தானியங்கிச் சோதனை (Automation Testing) பற்றித் திருமதி. து. நித்யா அவர்களின் மின் நூல் - கணியத்திலேயே கிடைக்கிறது - அதைப் படிப்பதற்கு முன்பு கொஞ்சம் நிரலாக்கம் படித்து விட்டு அந்தப் புத்தகத்தைப் படிப்பது கட்டாயத் தேவை. எப்படி சாப்ட்வேர் டெஸ்டிங்கை அடைய நினைப்பவர்களுக்கு அது எளிமையாகத் தெரிகிறதோ, அதே போல் தான் நிரலாக்கமும்! நிரலாக்கத்தைப் பொருத்த வரை பயிற்சி தான் தேவை. எனவே, கொஞ்சம் கூடுதல் நேரம் ஒதுக்கி அதையும் கைக்கொள்ளுங்கள். மென்பொருள் துறையில் வெற்றிக் கொடி நாட்டுங்கள்.