36.Peer Review for Test Cases and Test Plans: Implement and practice a peer review process for test cases and test plans to improve quality and identify potential issues early  
  
#include <stdio.h>

#include <string.h>

struct TestCase {

int id;

char description[100];

char expectedResult[50];

char status[20]; // Approved / Needs Improvement

char comments[100];

};

int main() {

int n, i;

printf("Enter number of test cases to review: ");

scanf("%d", &n);

struct TestCase testCases[n];

// Input test cases

for(i = 0; i < n; i++) {

testCases[i].id = i + 1;

getchar(); // clear buffer

printf("\nEnter description for Test Case %d: ", i + 1);

fgets(testCases[i].description, sizeof(testCases[i].description), stdin);

printf("Enter expected result for Test Case %d: ", i + 1);

fgets(testCases[i].expectedResult, sizeof(testCases[i].expectedResult), stdin);

strcpy(testCases[i].status, "Pending");

strcpy(testCases[i].comments, "Not Reviewed");

}

// Peer review process

printf("\n--- Peer Review Process ---\n");

for(i = 0; i < n; i++) {

int choice;

printf("\nReviewing Test Case %d\n", testCases[i].id);

printf("Description: %s", testCases[i].description);

printf("Expected Result: %s", testCases[i].expectedResult);

printf("Enter review decision (1 = Approve, 2 = Needs Improvement): ");

scanf("%d", &choice);

getchar(); // clear buffer

if(choice == 1) {

strcpy(testCases[i].status, "Approved");

strcpy(testCases[i].comments, "Looks good");

} else {

strcpy(testCases[i].status, "Needs Improvement");

printf("Enter reviewer comments: ");

fgets(testCases[i].comments, sizeof(testCases[i].comments), stdin);

}

}

// Display review summary

printf("\n--- Review Summary ---\n");

for(i = 0; i < n; i++) {

printf("\nTest Case %d\n", testCases[i].id);

printf("Description: %s", testCases[i].description);

printf("Expected Result: %s", testCases[i].expectedResult);

printf("Review Status: %s\n", testCases[i].status);

printf("Reviewer Comments: %s\n", testCases[i].comments);

}

return 0;

}

37.Defect Tracking with Tools: Use tools like Jira or Bugzilla to report testing progress and track defect resolution effectively.  
  
#include <stdio.h>

#include <string.h>

struct Defect {

int id;

char description[100];

char severity[20]; // Low, Medium, High

char status[20]; // New, In Progress, Resolved, Closed

char assignedTo[50];

};

int main() {

struct Defect defects[50];

int choice, count = 0, i, id;

while (1) {

printf("\n--- Defect Tracking System ---\n");

printf("1. Log New Defect\n");

printf("2. Update Defect Status\n");

printf("3. View All Defects\n");

printf("4. Exit\n");

printf("Enter choice: ");

scanf("%d", &choice);

getchar(); // clear buffer

if (choice == 1) {

defects[count].id = count + 1;

printf("Enter defect description: ");

fgets(defects[count].description, sizeof(defects[count].description), stdin);

printf("Enter severity (Low/Medium/High): ");

fgets(defects[count].severity, sizeof(defects[count].severity), stdin);

printf("Assign to developer: ");

fgets(defects[count].assignedTo, sizeof(defects[count].assignedTo), stdin);

strcpy(defects[count].status, "New");

printf("Defect logged successfully with ID: %d\n", defects[count].id);

count++;

}

else if (choice == 2) {

printf("Enter defect ID to update: ");

scanf("%d", &id);

getchar();

if (id <= 0 || id > count) {

printf("Invalid Defect ID!\n");

continue;

}

printf("Current Status: %s\n", defects[id - 1].status);

printf("Enter new status (New/In Progress/Resolved/Closed): ");

fgets(defects[id - 1].status, sizeof(defects[id - 1].status), stdin);

printf("Defect ID %d status updated successfully!\n", id);

}

else if (choice == 3) {

printf("\n--- Defect List ---\n");

for (i = 0; i < count; i++) {

printf("\nDefect ID: %d\n", defects[i].id);

printf("Description: %s", defects[i].description);

printf("Severity: %s", defects[i].severity);

printf("Assigned To: %s", defects[i].assignedTo);

printf("Status: %s\n", defects[i].status);

}

if (count == 0)

printf("No defects logged yet.\n");

}

else if (choice == 4) {

printf("Exiting Defect Tracking System...\n");

break;

}

else {

printf("Invalid choice! Try again.\n");

}

}

return 0;

}

38.Implement version control and configuration management for test scripts and environments.  
  
#include <stdio.h>

#include <string.h>

#define MAX\_SCRIPTS 30

#define MAX\_VERSIONS 20

#define MAX\_CONFIG\_KV 50

/\* ------------ Data Models ------------ \*/

typedef struct {

char key[32];

char value[128];

} KVPair;

typedef struct {

int version;

char message[120];

char content[512]; // test script content (or path)

} ScriptVersion;

typedef struct {

char name[64]; // unique script name

int versionCount;

ScriptVersion versions[MAX\_VERSIONS];

} Script;

typedef struct {

int version;

char message[120];

KVPair kv[MAX\_CONFIG\_KV];

int kvCount;

} ConfigSnapshot;

typedef struct {

Script scripts[MAX\_SCRIPTS];

int scriptCount;

ConfigSnapshot configs[MAX\_VERSIONS];

int configCount;

} Repo;

/\* ------------ Helpers ------------ \*/

void trim\_newline(char \*s){

size\_t n = strlen(s);

if(n && s[n-1]=='\n') s[n-1]=0;

}

int find\_script(Repo \*repo, const char \*name){

int i=0;

while(i<repo->scriptCount){

if(strcmp(repo->scripts[i].name, name)==0) return i;

i++;

}

return -1;

}

void print\_script(Script \*s){

if(s->versionCount==0){

printf(" (no versions)\n");

return;

}

ScriptVersion \*v = &s->versions[s->versionCount-1];

printf(" %s @ v%d | \"%s\"\n", s->name, v->version, v->message);

}

void list\_scripts(Repo \*repo){

if(repo->scriptCount==0){

printf("No scripts yet.\n");

return;

}

int i=0;

while(i<repo->scriptCount){

print\_script(&repo->scripts[i]);

i++;

}

}

void show\_script\_history(Repo \*repo){

char name[64];

printf("Enter script name: ");

fgets(name, sizeof(name), stdin); trim\_newline(name);

int idx = find\_script(repo, name);

if(idx<0){

printf("Script not found.\n");

return;

}

Script \*s = &repo->scripts[idx];

if(s->versionCount==0){

printf("No versions.\n");

return;

}

int i=0;

while(i<s->versionCount){

printf("v%-3d msg: %s\n", s->versions[i].version, s->versions[i].message);

i++;

}

}

void add\_script(Repo \*repo){

if(repo->scriptCount>=MAX\_SCRIPTS){

printf("Script limit reached.\n");

return;

}

char name[64], content[512], msg[120];

printf("New script name: ");

fgets(name, sizeof(name), stdin); trim\_newline(name);

if(find\_script(repo, name)>=0){

printf("Script already exists.\n");

return;

}

printf("Initial content/path: ");

fgets(content, sizeof(content), stdin); trim\_newline(content);

printf("Commit message: ");

fgets(msg, sizeof(msg), stdin); trim\_newline(msg);

Script \*s = &repo->scripts[repo->scriptCount];

strcpy(s->name, name);

s->versionCount = 1;

s->versions[0].version = 1;

strcpy(s->versions[0].content, content);

strcpy(s->versions[0].message, msg);

repo->scriptCount++;

printf("Script '%s' added at v1.\n", name);

}

void update\_script(Repo \*repo){

char name[64], content[512], msg[120];

printf("Script name to update: ");

fgets(name, sizeof(name), stdin); trim\_newline(name);

int idx = find\_script(repo, name);

if(idx<0){

printf("Script not found.\n");

return;

}

Script \*s = &repo->scripts[idx];

if(s->versionCount>=MAX\_VERSIONS){

printf("Version limit reached.\n");

return;

}

printf("New content/path: ");

fgets(content, sizeof(content), stdin); trim\_newline(content);

printf("Commit message: ");

fgets(msg, sizeof(msg), stdin); trim\_newline(msg);

int v = s->versionCount;

s->versions[v].version = v+1;

strcpy(s->versions[v].content, content);

strcpy(s->versions[v].message, msg);

s->versionCount++;

printf("Updated '%s' to v%d.\n", s->name, v+1);

}

void checkout\_script(Repo \*repo){

char name[64];

int target;

printf("Script name to checkout: ");

fgets(name, sizeof(name), stdin); trim\_newline(name);

int idx = find\_script(repo, name);

if(idx<0){

printf("Script not found.\n");

return;

}

Script \*s = &repo->scripts[idx];

if(s->versionCount==0){

printf("No versions.\n");

return;

}

printf("Target version (1..%d): ", s->versionCount);

if(scanf("%d", &target)!=1){ while(getchar()!='\n'); printf("Invalid input.\n"); return; }

while(getchar()!='\n');

if(target<1 || target> s->versionCount){

printf("Invalid version.\n");

return;

}

// Truncate history to selected version (like hard reset)

s->versionCount = target;

printf("Checked out '%s' to v%d.\n", s->name, target);

}

void show\_script\_current(Repo \*repo){

char name[64];

printf("Script name: ");

fgets(name, sizeof(name), stdin); trim\_newline(name);

int idx = find\_script(repo, name);

if(idx<0){

printf("Script not found.\n");

return;

}

Script \*s = &repo->scripts[idx];

if(s->versionCount==0){

printf("No versions.\n");

return;

}

ScriptVersion \*v = &s->versions[s->versionCount-1];

printf("Name: %s\nVersion: %d\nMessage: %s\nContent:\n%s\n",

s->name, v->version, v->message, v->content);

}

/\* ------------ Config Management ------------ \*/

void config\_set\_key(ConfigSnapshot \*snap){

if(snap->kvCount>=MAX\_CONFIG\_KV){

printf("KV capacity reached.\n");

return;

}

char key[32], val[128];

printf("Key: ");

fgets(key, sizeof(key), stdin); trim\_newline(key);

printf("Value: ");

fgets(val, sizeof(val), stdin); trim\_newline(val);

// Upsert

int i=0, found=-1;

while(i<snap->kvCount){

if(strcmp(snap->kv[i].key, key)==0) found=i;

i++;

}

if(found>=0){

strcpy(snap->kv[found].value, val);

printf("Updated %s.\n", key);

} else {

strcpy(snap->kv[snap->kvCount].key, key);

strcpy(snap->kv[snap->kvCount].value, val);

snap->kvCount++;

printf("Added %s.\n", key);

}

}

void show\_working\_config(ConfigSnapshot \*snap){

if(snap->kvCount==0){

printf("(empty)\n");

return;

}

int i=0;

while(i<snap->kvCount){

printf("%s = %s\n", snap->kv[i].key, snap->kv[i].value);

i++;

}

}

void commit\_config(Repo \*repo, ConfigSnapshot \*working){

if(repo->configCount>=MAX\_VERSIONS){

printf("Config version limit reached.\n");

return;

}

char msg[120];

printf("Commit message: ");

fgets(msg, sizeof(msg), stdin); trim\_newline(msg);

ConfigSnapshot \*dst = &repo->configs[repo->configCount];

dst->version = repo->configCount + 1;

strcpy(dst->message, msg);

dst->kvCount = working->kvCount;

int i=0;

while(i<working->kvCount){

dst->kv[i] = working->kv[i];

i++;

}

repo->configCount++;

printf("Committed config snapshot v%d.\n", dst->version);

}

void show\_config\_history(Repo \*repo){

if(repo->configCount==0){

printf("No config commits.\n");

return;

}

int i=0;

while(i<repo->configCount){

printf("v%-3d msg: %s (entries: %d)\n",

repo->configs[i].version, repo->configs[i].message, repo->configs[i].kvCount);

i++;

}

}

void checkout\_config(Repo \*repo, ConfigSnapshot \*working){

if(repo->configCount==0){

printf("No config commits to checkout.\n");

return;

}

int t;

printf("Target config version (1..%d): ", repo->configCount);

if(scanf("%d", &t)!=1){ while(getchar()!='\n'); printf("Invalid input.\n"); return; }

while(getchar()!='\n');

if(t<1 || t>repo->configCount){

printf("Invalid version.\n");

return;

}

ConfigSnapshot \*src = &repo->configs[t-1];

working->kvCount = src->kvCount;

int i=0;

while(i<src->kvCount){

working->kv[i] = src->kv[i];

i++;

}

printf("Checked out config to v%d (not a new commit; working copy updated).\n", t);

}

/\* ------------ Main ------------ \*/

int main(){

Repo repo;

repo.scriptCount = 0;

repo.configCount = 0;

// Working (uncommitted) environment config

ConfigSnapshot working;

working.version = 0;

strcpy(working.message, "working");

working.kvCount = 0;

int choice = 0;

int running = 1;

while(running){

printf("\n=== Test Assets VC & Config Manager ===\n");

printf("1 Add script\n");

printf("2 Update script (new version)\n");

printf("3 Show script history\n");

printf("4 Checkout script version (hard reset)\n");

printf("5 Show current script content\n");

printf("6 Config: set/update key\n");

printf("7 Config: show working copy\n");

printf("8 Config: commit snapshot\n");

printf("9 Config: history\n");

printf("10 Config: checkout snapshot to working\n");

printf("11 List scripts (summary)\n");

printf("12 Exit\n");

printf("Enter choice: ");

if(scanf("%d", &choice)!=1){

while(getchar()!='\n');

printf("Invalid input.\n");

} else {

while(getchar()!='\n'); // clear line

if(choice==1) add\_script(&repo);

else if(choice==2) update\_script(&repo);

else if(choice==3) show\_script\_history(&repo);

else if(choice==4) checkout\_script(&repo);

else if(choice==5) show\_script\_current(&repo);

else if(choice==6) config\_set\_key(&working);

else if(choice==7) show\_working\_config(&working);

else if(choice==8) commit\_config(&repo, &working);

else if(choice==9) show\_config\_history(&repo);

else if(choice==10) checkout\_config(&repo, &working);

else if(choice==11) list\_scripts(&repo);

else if(choice==12) running = 0;

else printf("Invalid choice.\n");

}

}

printf("Goodbye.\n");

return 0;

}

39.Use tools like Git, SVN, or Mercurial for managing test scripts, defect logs, and test results.  
  
/\* vc\_manager\_git.c

Simple Git wrapper to manage test scripts, defect logs, and test results.

Compile: gcc -o vc\_manager\_git vc\_manager\_git.c

Run: ./vc\_manager\_git

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define CMD\_BUF 1024

#define PATH\_MAX\_LEN 512

void run\_cmd(const char \*cmd) {

int rc = system(cmd);

if (rc != 0) {

printf("[WARNING] Command failed (exit %d): %s\n", rc, cmd);

}

}

void run\_cmd\_capture(const char \*cmd) {

char buf[512];

FILE \*fp = popen(cmd, "r");

if (!fp) {

printf("Failed to run command: %s\n", cmd);

return;

}

while (fgets(buf, sizeof(buf), fp)) {

fputs(buf, stdout);

}

pclose(fp);

}

void init\_repo() {

printf("Initializing git repository (if not already)...\n");

run\_cmd("git rev-parse --is-inside-work-tree > /dev/null 2>&1 || git init");

}

void add\_and\_commit() {

char msg[256];

printf("Enter commit message: ");

if (!fgets(msg, sizeof(msg), stdin)) return;

msg[strcspn(msg, "\n")] = 0; // trim newline

// Stage common artifact directories if they exist

run\_cmd("test -d tests && git add tests || true");

run\_cmd("test -d defects && git add defects || true");

run\_cmd("test -d results && git add results || true");

// If user wants to add arbitrary paths

char extra[PATH\_MAX\_LEN];

printf("Add extra path (or press Enter to skip): ");

if (!fgets(extra, sizeof(extra), stdin)) return;

extra[strcspn(extra, "\n")] = 0;

if (strlen(extra) > 0) {

char cmd[CMD\_BUF];

snprintf(cmd, sizeof(cmd), "git add \"%s\"", extra);

run\_cmd(cmd);

}

char cmd[CMD\_BUF];

snprintf(cmd, sizeof(cmd), "git commit -m \"%s\" || echo \"Nothing to commit or commit failed\"", msg);

run\_cmd(cmd);

}

void create\_branch() {

char branch[128];

printf("Enter branch name to create/switch to: ");

if (!fgets(branch, sizeof(branch), stdin)) return;

branch[strcspn(branch, "\n")] = 0;

if (strlen(branch) == 0) return;

char cmd[CMD\_BUF];

snprintf(cmd, sizeof(cmd), "git checkout -b \"%s\" 2>/dev/null || git checkout \"%s\"", branch, branch);

run\_cmd(cmd);

printf("Now on branch '%s'\n", branch);

}

void create\_tag() {

char tag[128], msg[256];

printf("Enter tag name: ");

if (!fgets(tag, sizeof(tag), stdin)) return;

tag[strcspn(tag, "\n")] = 0;

if (strlen(tag) == 0) return;

printf("Enter tag message (optional): ");

if (!fgets(msg, sizeof(msg), stdin)) return;

msg[strcspn(msg, "\n")] = 0;

char cmd[CMD\_BUF];

if (strlen(msg) > 0) {

snprintf(cmd, sizeof(cmd), "git tag -a \"%s\" -m \"%s\" || echo \"tag failed\"", tag, msg);

} else {

snprintf(cmd, sizeof(cmd), "git tag \"%s\" || echo \"tag failed\"", tag);

}

run\_cmd(cmd);

printf("Created tag '%s'\n", tag);

}

void show\_log() {

printf("Recent commits (git log --oneline -n 20):\n");

run\_cmd\_capture("git --no-pager log --oneline -n 20");

}

void push\_remote() {

char remote[256], branch[128];

printf("Enter remote name (e.g., origin) or press Enter to skip: ");

if (!fgets(remote, sizeof(remote), stdin)) return;

remote[strcspn(remote, "\n")] = 0;

if (strlen(remote) == 0) return;

printf("Enter branch to push (or press Enter to push current branch): ");

if (!fgets(branch, sizeof(branch), stdin)) return;

branch[strcspn(branch, "\n")] = 0;

char cmd[CMD\_BUF];

if (strlen(branch) == 0) {

snprintf(cmd, sizeof(cmd), "git push %s", remote);

} else {

snprintf(cmd, sizeof(cmd), "git push %s %s", remote, branch);

}

run\_cmd(cmd);

}

int main() {

int choice = 0;

while (1) {

printf("\n=== Test Assets Git Manager ===\n");

printf("1. Init repository\n");

printf("2. Add & commit (tests/ defects/ results/ or extra)\n");

printf("3. Create/switch branch\n");

printf("4. Create tag\n");

printf("5. Show recent log\n");

printf("6. Push to remote\n");

printf("7. Exit\n");

printf("Enter choice: ");

if (scanf("%d", &choice) != 1) {

while (getchar() != '\n');

printf("Invalid input\n");

continue;

}

while (getchar() != '\n'); // flush newline

if (choice == 1) init\_repo();

else if (choice == 2) add\_and\_commit();

else if (choice == 3) create\_branch();

else if (choice == 4) create\_tag();

else if (choice == 5) show\_log();

else if (choice == 6) push\_remote();

else if (choice == 7) break;

else printf("Invalid choice\n");

}

printf("Exiting.\n");

return 0;

}

40.Perform test closure activities, including documentation, analysis, and final reporting for a project.  
  
#include <stdio.h>

struct TestClosure {

int totalTestCases;

int passed;

int failed;

int defectsLogged;

char projectName[50];

char closureDate[20];

};

void generateClosureReport(struct TestClosure t) {

printf("\n===== TEST CLOSURE REPORT =====\n");

printf("Project Name : %s\n", t.projectName);

printf("Closure Date : %s\n", t.closureDate);

printf("Total Test Cases Executed : %d\n", t.totalTestCases);

printf("Passed Test Cases : %d\n", t.passed);

printf("Failed Test Cases : %d\n", t.failed);

printf("Defects Logged : %d\n", t.defectsLogged);

// Simple analysis

float passRate = ((float)t.passed / t.totalTestCases) \* 100;

float failRate = ((float)t.failed / t.totalTestCases) \* 100;

printf("\n--- Analysis ---\n");

printf("Pass Percentage : %.2f%%\n", passRate);

printf("Fail Percentage : %.2f%%\n", failRate);

if (failRate > 20) {

printf("Conclusion: High failure rate. More testing needed before release.\n");

} else {

printf("Conclusion: Testing completed successfully. Ready for closure.\n");

}

printf("===== END OF REPORT =====\n");

}

int main() {

struct TestClosure t;

printf("Enter Project Name: ");

scanf(" %[^\n]", t.projectName);

printf("Enter Closure Date (dd-mm-yyyy): ");

scanf(" %[^\n]", t.closureDate);

printf("Enter Total Test Cases Executed: ");

scanf("%d", &t.totalTestCases);

printf("Enter Number of Passed Test Cases: ");

scanf("%d", &t.passed);

printf("Enter Number of Failed Test Cases: ");

scanf("%d", &t.failed);

printf("Enter Number of Defects Logged: ");

scanf("%d", &t.defectsLogged);

// Generate the final test closure report

generateClosureReport(t);

return 0;

}