



## ניפוי שגיאות

### תמי לפידות

"מגוון - מחקר ופיתוח בהוראת מדעי המחשב"  
הוראת הטכנולוגיה והמדעים, טכניון

אולם, למרות הנסיונות השונים לפתח מערכות מסוג זה, לא צפויה בעתיד מערכת מושלמת שתזהה את כל סוגי השגיאות ותתקן אותן. מערכת "המנפה האוטומטי" לא ניתנת לבנייה בעיקר בשל הבעיות הכרוכות בגילוי מראש של שגיאות הרצה וזיהוי של שגיאות לוגיות. לפיכך, נותרה מלאכת ניפוי השגיאות באחריותו הכמעט בלעדית של המתכנת.

#### מקור המונח debugging

בז'רגון המקצועי, נהוג לקרוא לשגיאת תכנות בשם bug, ולפעולה של איתור ותיקון ה-bug, נהוג לקרוא בשם debugging. מהו מקור השם? לפי Patis, אחד הסיפורים מייחס את מקור המונח למהנדסי חברת הטלפון. הם השתמשו בו כדי לתאר את המקור לרעשים אקראיים שהתגלו במעגלי התקשורת האלקטרוניים. "אגדת-עם" פופולרית יותר, מספרת שמקור השם הוא בתקופת המחשבים הראשונים. במחשב Mark-I (אוניברסיטת הרוורד) התקבלו תוצאות שגויות. לאחר שפרקו אותו למרכיביו, כדי לנסות ולגלות את התקלה, מצאו חרק (באנגלית bug) מת שגרם לקצר במעגלים. Patis טוען שהמושג bug בא כדי להגן על ה"אגו" של המתכנתים, שלא יכלו להודות באמת הפשוטה שתכניותיהם מלאות בשגיאות. לדעתו, המטפורה די מוצלחת כיון ש-bug-ים הם קטנים וקשה למצאם. כמו כן, למרות שניתן למצוא bug מסוים ולסלקו, לא מובטח לנו שלא הסתננו bug-ים נוספים.

#### מבוא

אחת הפעילויות המעסיקות את המתכנת חלק ניכר מזמנו, היא ניפוי שגיאות. כלומר, איתור שגיאות בתכניות שאינן מתנהגות כראוי ותיקונן לשביעות רצונו. זהו גם אחד מהנושאים הקשים ביותר להוראה. ננסה לבחון את הקשיים בהוראת הנושא ומספר שיטות להתמודד איתם.

קורס שבמרכזו עומדת שפת תיכנות והשימוש בה לצורך פתרון בעיות, חייב לטפל גם בעובדת קיומן הבלתי-נמנע של שגיאות. כל מתכנת יודע שלא ניתן למנוע "הסתננות" של שגיאות לתוכנית.

בשלבם הראשונים של התפתחות שפות התכנות ולפני פיתוחן של שפות עיליות, מתכנתים נאלצו לקרוא memory dumps שהכילו כתובות בזכרון ותכולתן. הם נאלצו לשחזר את פעולות התכנית ולאתר את שגיאותיהם במאמץ רב. אולם כבר אז, התחילו לפתח אסטרטגיות לניפוי שגיאות (כמו tracing והדפסות ביקורת). במשך השנים התפתחו אסטרטגיות שונות ופותחו כלים שיסייעו בתהליך המורכב של ניפוי השגיאות.

בשנים האחרונות, נעשו מספר נסיונות לבנות מערכות מחשב (בדרך כלל מבוססות על אינטליגנציה מלאכותית) שיאתרו שגיאות בתכניות. לדוגמה, מערכת HACKER המנסה לזהות שגיאות בסביבה של "עולם הקוביות", או מערכת PROUST שמזהה שגיאות לא תחביריות בעזרת השוואה בין מטרות התכנית לבין הסכמות המיוצגות בה.

כלי-עזר שפיתחו ואימצו המומחים מתוך נסיונם המצטבר.

Soloway ו-Spohrer -1 מציינים שניתן לחלק את העניין האקדמי בניפוי שגיאות לפי מספר תחומים: מהנדסים מעוניינים לשפר את הפרודוקטיביות של המתכנתים (מחקרים הראו ש-70% מזמן הפיתוח של תכנה, מוקדש לניפוי) ואת מהימנות התכנות שלהם; מחנכים מעוניינים לשפר את הבנתם של מתחילים, הן מבחינת תהליכי התכנות והן מבחינת יכולתם ללמוד במהרה כיצד לכתוב תכנות נכונות; ואילו אנשי המדע הקוגניטיבי מעוניינים לשפר את הבנתנו בנושאי הרכישה והביצוע של מיומנויות הכרוכות בפתרון בעיות מורכבות. כמו כן, הם מעוניינים לקבוע כיצד תפיסות מוטעות, אסטרטגיות של פתרון בעיות ואילוצים קוגניטיביים שונים יכולים להוביל לשגיאות.

החשיבות החינוכית של העיסוק בניפוי שגיאות מתמקדת בגורמים הבאים:

- שינוי העמדות השליליות של התלמידים כלפי שגיאות וחשיבותן בתהליך הלמידה. "הפילוסופיה של ניפוי השגיאות גורסת שהשגיאות והעיסוק בהן מחכימים אותנו, כיון שהם מביאים אותנו לבדוק מה קרה, להבין מה השתבש ובאמצעות ההבנה הזאת, להצליח בתיקון השגיאה.
- שיפור ההבנה של התלמידים לגבי תהליך החשיבה שלהם. כתוצאה מכך ששגיאות של תלמידים הופכות לנושא לגיטימי לשיחה בכתה, ניתן לטפל בתהליכי חשיבה ובתפיסות מוטעות.
- תהליך התכנות מסייע בהקניית מיומנויות של פתרון בעיות. הנסיון שצוברים התלמידים במהלך התכנות, מביא אותם "להאמין" בניפוי שגיאות שהוא מקרה פרטי של פתרון בעיות.

Papert סבור שלפעולת התכנות ולעיסוק בניפוי שגיאות יש פוטנציאל להשפיע על התפיסה המקובלת של שגיאות בתהליך הלמידה. לומדים חוששים מעשיית שגיאות ונוטים לתפוס את ההצלחות או הכשלונות שלהם בראייה של "שחור

לעומתו, Dijkstra (1988) תוקף את השימוש במטפורה של bug וטוען שיש להשתמש במונח error. הוא טוען שהשימוש במושג bug, יוצר רושם מטעה לפיו השגיאות נכנסות לתכנית באקראי, ללא שליטת המתכנת. ולפיכך, יש הנוטים לדבר על תכניות "כמעט נכונות". לעומת זאת, אם נשתמש ב"שגיאה", הרי שהאשמה היא בפירוש במתכנת וכאשר התכנית מכילה ולו שגיאה בודדת יחידה, הרי היא פשוט "שגויה". הוא אף מרחיק ומציע, בהומור, לקנוס כל איש אקדמיה שישתמש במושג bug ובכך, לדעתו, יפתרו בעיות התקציב הקשות של האוניברסיטאות.

### חשיבות הטיפול בניפוי שגיאות במהלך ההוראה

מוסכם על העוסקים במלאכת התכנות, שניפוי הינו מלאכה לא קלה ואפילו מומחים יכולים להתקשות בה. אולם, למרות הקשיים, לא כדאי להמנע מהוראת ניפוי שגיאות לתלמידי תיכון. לא זאת בלבד, אלא שהנושא צריך ללוות את התלמיד במשך כל הקורס. ראשית, מובן מאליו שאין ברירה, אי אפשר לתכנת ללא שגיאות. לפיכך, כדי ללמוד תכנות, חייבים לדעת גם איך לנפות שגיאות. יתר על כן, לעיסוק בניפוי שגיאות יש תועלת חינוכית מיוחדת. כאמור, לא ניתן לפתח מערכת ממוחשבת אוטומטית שתזהה ותתקן את כל השגיאות הקיימות בתכנית. אולם אפילו אם היתה קיימת תוכנה מושלמת מעין זו, היה ראוי מבחינה חינוכית להתעכב על הוראת האסטרטגיות השונות לניפוי שגיאות.

הבעיות הכרוכות בניפוי שגיאות, מעסיקות את אנשי המקצוע ולא ניתן להתחמק מהן גם בשלבים הראשונים והפשוטים ביותר של לימוד התיכנות. יחד עם זאת זהו אחד התחומים החשובים, הואיל ופיתוחה של תכנית נכונה הוא למעשה תכלית פעולת התיכנות.

מאחר שדוקא מתחילים, שאינם שולטים עדיין ברזי השפה ובשיטות העבודה, צפויים להיתקל בשגיאות בתכיפות רבה, רצוי לסייע להם בעזרת הצגה של

עד שמגיעים למודל הנכון או לתכנית הרצויה. כאשר תלמידים עוסקים בבניית תכניות ובניפוי, הם מתוודעים לדרכי הלמידה של עצמם ויש סיכוי שיכירו טוב יותר את אופן החשיבה שלהם.

גם Soloway ו-Spohrer רואים חשיבות מיוחדת בעיסוק בשגיאות, כיוון שהן "פותחות לנו צוהר להתבוננות במושגים המנטליים שיצר המתכנת בראשו". Guilo ו-Clements אף מצאו שהעיסוק בניפוי שגיאות משפיע על סגנון פתרון הבעיות של תלמידים ומביא אותם להפעלת יותר רפלקציה וחשיבה עצמית על עבודתם. ונסיים בדברים של Carver ו-McCoy: "אחד הצידוקים המרכזיים להוראת תכנות בתיכון הוא בכך שזו הזדמנות מצוינת להציג לתלמידים מיומנויות ברמה גבוהה, כמו ניפוי שגיאות. יחד עם זאת, לא ניתן לצפות שתלמידים ירכשו את המיומנויות האלו, אלא אם כן, הם ילמדו אותן במפורש".

### מדוע קשה להמנע משגיאות?

ניתן לראות ניפוי שגיאות כמקרה פרטי של פתרון בעיות. אולם, מדוע קשה כל-כך לכתוב תכניות ללא שגיאות ומהם הקשיים המייחדים ניפוי שגיאות מכל תהליך אחר של פתרון בעיות?

ראשית, לא ניתן להמנע משגיאות, כיון שבין היתר, לא ניתן לחזות מראש את כל הבעיות שיתעוררו במהלך התכנות. Pattis מזהיר את המתכנת מפני הסינדרום של "אני חייב להצליח בפעם הראשונה".

הוא מציע למתכנת לא לנסות אפילו להמנע משגיאות, כיון שלא ניתן לצפות מראש את כל הבעיות שיתעוררו במהלך כתיבת התכנית. כמו כן, גישה כזאת יכולה להביא את המתכנת למבוי סתום ולחוסר היכולת להתחיל בכתיבה.

יתר על כן, לפי הגישה הקונסטרוקטיביסטית, תהליך הלמידה נתפס כתהליך של ניפוי ושיפור מודלים שגויים. כלומר, כל למידה של מושג או רעיון חדש מתחילה מהשלב ההכרחי של בניית מודל שגוי. במשך הזמן, המודל יתוקן וישופר עד לקבלת מודל המתאים יותר למציאות.

לבן". החברה המערבית המודרנית נוטה לראות בהצלחה מטרה עיקרית וכל שגיאה נתפסת כדבר שלילי ומפריע בתהליך הקידמה. השקפה זו מתחזקת בקרב תלמידי תיכון שהורגלו להישפט על השגיהם על-ידי מערכת של מבחנים וציונים. המסר החינוכי הסמוי שהועבר להם במשך השנים היה: טעות או שגיאה תגרע מציוניך וככל שתמנע מטעויות הרי שציוניך יעלו. תלמיד שמצליח להמנע משגיאות, הוא תלמיד מצטיין ו"חכם" ותלמיד ששוגה באופן תכוף, הוא חלש ו"טיפש". התפיסה השלילית הזו כלפי שגיאות היא אחת הבעיות הקשות בהוראת מדעי המחשב.

תפקיד המורה צריך להתפס, בהקשר זה, במובן הכללי יותר של שינוי עמדות ומוסכם שזה אינו תפקיד קל. Pattis מנסה להתמודד עם האתגר החינוכי הזה: בדרך כלל, קשה למצוא שגיאות תכנות ולאחר שמוצאים אותן הן נראות לעיתים קרובות מובנות-מאליהן. אל תכעס על עצמך כאשר אתה מוצא שגיאות כאלו. אם אתה מוצא שגיאה "טפשית", אל תחשוב "אני ממש אדיוט"; במקום לתפוס זאת ככשלון, חשוב "אני צריך להיות גאון כדי למצוא שגיאה כל-כך מוזרה וקשה לאיתור".

כמובן שאין צורך לעבוד עם מחשבים כדי לשגות, ומובן שאסטרטגיות לניפוי שגיאות פותחו על-ידי לומדים גם לפני שנוצר המחשב הראשון. אולם העיסוק בתכנות מחدد את הנושא ומאפשר לנו לעמוד ביתר בהירות על אסטרטגיות החשיבה שאנו נוקטים.

לפי Papert, כל תכנית שאנו כותבים, היא מעין מודל שמשקף כיצד אנו תופסים את הבעיה שעלינו לפתור. כל משימת תיכנות משקפת נסיון שלנו להתקרב טוב יותר להבנת העולם בו אנו חיים. כאשר אנו כותבים תכנית שגויה ומתקנים אותה, אנו מתקרבים לקראת הבנה טובה יותר של המציאות. תהליך הלמידה של נושא כלשהו, מתחיל מבניית מודל ראשוני (שגוי בדרך-כלל). ההכרה שהמודל שגוי, היא חיונית בתהליך הלמידה כדי שיתעורר הצורך בשכתוב המודל (התכנית) ושיפורו. רק לאחר שלבים חוזרים של ניפוי שגיאות מהמודל, אנו מתקרבים בהדרגה למודל משופר ומתאים יותר,

בשפת החשיבה האנושית ובשפה הטבעית, דרך שפות ביניים לתכנון הפתרון (לדוגמה, פסאודו-קוד או תרשימי זרימה), ועד לשפת התכנות הגבוהה והשלכותיה ברמת שפת המכונה.

גם אם לא מקבלים את עמדתו הקיצונית של Dijkstra, ברור ששגיאות הן בלתי נמנעות וניפוין הוא מלאכה מורכבת ובעייתית. יחד עם זאת, העיסוק בהם הוא בעל ערך חינוכי רב. לפיכך, ראוי להקדיש זמן ומחשבה איך ניתן לצייד את התלמידים בכלים מתאימים להתמודדות עם הנושא ואף להפיק מהתמודדות זו תועלת חינוכית.

### סיווג שגיאות

את השגיאות הנוצרות בתהליך התיכנות, ניתן לסווג לפי בסיסי מיון שונים.

הראל, למשל, מציע את החלוקה הבאה:

- שגיאות שפה (language error) קשורות בחריגה מכללי התחביר של השפה. הן מתגלות ע"י המערכת וניתנות לתיקון במאמץ מועט יחסית.
- שגיאות לוגיות (logical error) נובעות מאלגוריתם שגוי לפתרון הבעיה. הראל מחלק אותן לשגיאות סמנטיות ואלגוריתמיות.
- שגיאות סמנטיות נובעות מסיבות שאינן מורות על חשיבה שגויה. לדוגמה: חוסר הבנה של הסמנטיקה של שפת התיכנות. או אי-טיפול במקרי גבול, כמו אפסים או עלים.
- שגיאות אלגוריתמיות נובעות משגיאות בלוגיקה של תכנון הפתרון. לפעמים, אם ישחק לנו המזל, שגיאה כזו תתגלה כאשר המערכת לא תוכל לבצע הוראה מסוימת. ברוב המקרים, השגיאה לא תתגלה ע"י המערכת ולפיכך, אלו הן השגיאות הקשות ביותר לאיתור ולניפוי. הן גם הקשות ביותר להוראה כיון שיש כאן קשר ישיר לתהליכים קוגניטיביים מורכבים של חשיבה ולמידה, שהם לרוב סמויים מן העין.

אפשר למיין שגיאות לבי בסיסי מיון אחרים. למשל: שגיאות מילון (lexical) הנגרמות כאשר נותנים

Perkins, Schwartz, Simmons מציינים את הקשר בין תיכנות ותהליכי החשיבה הנדרשים לצורך ביצועו. הם טוענים שלא מדובר בידע מושלם או בחוסר ידע מוחלט. בדרך כלל, לתלמיד יש מושג מסוים על המשימה שעליו לבצע, הוא "מבין בערך" או "יודע קצת". גם Perkins ו-Martin דנים בידע החלקי שיש לתלמיד. הם מכנים זאת בשם "ידע שביר" (fragile knowledge) ומחלקים אותו לארבעה סוגי ידע:

- ידע חסר (missing knowledge), כאשר לתלמיד חסר ידע על הוראה מסוימת או על עקרון תיכנותי מסוים, כיון שלא למד אותם עדיין;
- ידע "סביל" (inert knowledge), כאשר הידע קיים אצל התלמיד, אולם הוא לא מצליח ל"שלוף" אותו לצורך שימוש ברגע המתאים;
- ידע לא רלבנטי (misplaced knowledge), כאשר תלמיד מיישם ידע שרכש, אך לא במקום המתאים;
- וידע "מגובב" (conglomerated knowledge), כאשר תלמיד מנסה ליצור קשרים בין קטעי קוד שונים, אולם הוא עושה זאת באופן שגוי. במקרה זה, התלמיד יוצר מעין גיבוב של דברים ללא הקשר מתאים.

ה"ידע השביר" בא לידי ביטוי חריף יותר כאשר המתכנת אינו שולט באסטרטגיות בסיסיות של פתרון בעיות.

מספר חוקרים מאמינים שיש קושי מיוחד דוקא בתכנות לעומת מקרים אחרים של פתרון בעיות.

Perkins ו-Martin טוענים כי פעולת התיכנות דורשת סוג של שלמות שלא נדרש מהתלמידים במקצועות אחרים. במבחן רגיל, לדוגמה, אם תטעה ב- 10% מהתשובות, עדיין תוכל לקבל 90% שנחשב כציון טוב-מאד. לעומת זאת, אם תכתוב תכנית שהיא נכונה ב- 90%, התכנית לא תעבוד ודוקא 10% הנותרים ייצרו בעיות ושגיאות שקשה לנפותן.

Dijkstra טוען שהסיבה המכרעת לקיומן הבלתי נמנע של שגיאות, נובעת מייחודה של פעולת התיכנות, המחייבת את המתכנת עיסוק ברמות מרובות של הפשטה והבנה. לדבריו, המתכנת נאלץ להתמודד עם היררכיות מרובות של שפות. החל מניתוח הבעיה

ממוצע בבית-ספר תיכון, גם הן בעייתיות ובמקרים רבים, הוא אינו יודע אפילו לזהות אותן. לפיכך, גם העיסוק בשגיאות מסוג זה, נושא ערך חינוכי. הוא מאפשר לנו לטפל בעמדות של התלמידים כלפי שגיאות ולערוך להם הכרות ראשונה עם האסטרטגיות המקובלות לניפוי שגיאות.

### שגיאות שאינן תלויות-שפה

לפי כל החלוקות שהוזכרו, אלו הן השגיאות הקשות ביותר לאיתור וניפוי ולפיכך, גם הקשות ביותר להוראה. במקרה זה, יתכן שהמחשב לא יתן כל אינדיקציה לעצם קיומה של שגיאה, והאחריות לגילוייה וניפוייה מוטלת על המתכנת. כלומר, התלמיד נדרש להבין כי סיום הביצוע של התכנית ללא "תלונות" מצד המערכת, עדיין אינו מבטיח שהתכנית התבצעה על-פי התכנון והכוונות שלו.

יכולות להיות מספר סיבות להוצרות שגיאה לוגית: יתכן שהאלגוריתם שלנו שגוי ואז צריך לחזור לשלב התכנון, יתכן שהתרגום של האלגוריתם שגוי, יתכן שלא צפינו מקרים מסוימים (מקרי גבול), ויתכן שלא "תפרנו" נכון את המודולים של התכנית.

אולם לפני שניגשים לאתר את הסיבה לשגיאה, יש לגלות את קיומה או להבין שבכלל היתה שגיאה. במילים אחרות, הבעייה הראשונה הקשורה בשגיאות לוגיות, היא כיצד נוכל לדעת האם פתרנו את המשימה? או כיצד נדע שהתכנית שכתבנו אכן מבצעת את הנדרש? לשם כך, עלינו להרגיל את התלמידים להגדיר למה בדיוק הם מצפים כתוצאה מהפעלת התכנית; איזה תוצאות יהיו מקובלות עליהם כסבירות ואילו תוצאות "ידליקו אור אדום" כמדד להתנהגות לא תקינה של התכנית.

בהקשר זה, ראוי להזכיר את היתרון הרב שיש לעבודה בסביבה גרפית. כאשר התכנית השגויה גורמת לציור או שרטוט שונה מהמצופה, הרי שעצם הזיהוי של קיום השגיאה יכול לסייע לתלמיד כשלב ראשון בניפוי. בסביבות לא-גרפיות, קשה יותר לזהות שהתכנית לא ביצעה את המשימה, כיון שאז יש צורך לעקוב אחר הדפסות מורכבות או לבצע חישובים שיאשרו / יפריכו את נכונות הפתרון.

למחשב מילה שאינה נמצאת באוצר המילים שלו. שגיאות תחביר (syntactic) הנגרמות כאשר מפריס את חוקי התחביר של השפה. שגיאות ביצוע (execution) הנגרמות כאשר המחשב אינו מצליח לבצע בהצלחה הוראה מסוימת. ושגיאות כוונה (intent) הנגרמות כאשר המחשב מסיים את ביצוע התכנית, אולם הוא ביצע משהו שונה מהמשימה אליה התכוון המתכנת.

### שגיאות תלויות-שפה

שגיאות אלה ניתנות לגילוי באופן קל יחסית כיון שהמערכת מודיעה, בדרך כלל, על גילוי השגיאה, על מיקומה ובמקרים רבים אף ניתנים רמזים המסייעים בתיקון השגיאה. לפיכך גם קל יותר ללמד איך לנפותן. יש צורך בהכרות עם חוקי התחביר של השפה שבה כתובה התכנית ואפשר להסתייע ב"הרצה יבשה" (tracing).

מתכנת מיומן אף יכול להקטין את הופעתן של שגיאות התחביר, אם יקרא את תכניתו היטב לפני הפעלתה. כאשר עובדים בסביבת קומפילר, נהוג לכנות אותן בשם "שגיאות קומפילציה" כיון שהמערכת יכולה לזהות את השגיאות כבר בשלב הראשון של קריאת התכנית ותרגומה. למעשה, זהו הסוג היחיד שניתן לטפל בו בהצלחה מלאה בעזרת מערכת ממוחשבת אוטומטית.

שגיאות מילון ותחביר הן תלויות-שפה, כיון שלכל שפה יש מילון שונה וחוקי תחביר שונים. מה ששגוי בשפה אחת, יכול להחשב תקין בשפה אחרת. הגורמים שמביאים תלמיד להכנסת שגיאות תלויות-שפה לתכניתו, יכולים להיות חוסר תשומת לב לפרטים, חוסר הכרות מספיקה עם התחביר או חוסר הבנת הסמנטיקה של השפה. בכל אופן, כדי לסייע לתלמידים באיתור שגיאות אלו, עלינו להרגיל אותם לקרוא את הודעות המערכת, להנחותם כיצד להעזר ב-manual של השפה, ולתרגל אותם בהבנת המידע העומד לרשותם ויישומו לצורך הניפוי.

רוב המתכנתים המיומנים מתייחסים לשגיאות שהזכרנו כשגיאות טריביאליות. אולם, עבור תלמיד

שיש חשיבות רבה להודעות השגיאה של המערכת. אם ההודעה היא "RUN-TIME ERROR: CODE 72" יקשה מאד לטפל בה. לעומת זאת, יש מערכות שמספקות למשתמש הודעות מפורטות יותר המרמזות על מיקום השגיאה וכיצד לטפל בה.

בחלק מהמקרים, ניתן למנוע הופעתן של שגיאות ביצוע. לשם כך, יש לקחת בחשבון מקרי גבול, לבדוק היטב את הקשרים בין המודולים של התכנית, ולבנות מסננות קלט שיטפלו בכל המקרים האפשריים.

כדי לסייע לתלמידים באיתור שגיאות ביצוע ואולי אף במניעתן, מומלץ להקדיש זמן לנושא של מסננת-קלט. כמו כן, ראוי להדגים לתלמידים את החשיבות של בדיקת מקרי הגבול האופייניים לכל בעייה. (אם מקפידים על פיתוח הפתרון בעזרת עידון הדרגתי, הרי שהטיפול במקרי גבול בא לידי ביטוי באופן טבעי).

העובדה היא שיש שגיאות שקל יותר לאתר ויש שגיאות שקשה יותר לזהות ולנפות. בכל המקרים, לא ניתן להגדיר כללי התנהגות "נכונים" שימנעו הופעתן של שגיאות. לפיכך, קשה גם ללמד איך מזהים שגיאות ואיך להימנע מהן. זהו אחד מהתחומים "הרכים" של מדעי המחשב וכדי להתמודד איתו יש צורך בתרגול רב ובפיתוח אסטרטגיות שיסייעו לתלמיד.

### אסטרטגיות לניפוי שגיאות

האסטרטגיה היעילה ביותר להמנעות משגיאות או לניפוי, היא ללא ספק, שימוש בסגנון תכנות טוב וחשיבה בהירה ומסודרת בכל שלבי העבודה. יחד עם זאת, תלמידי תיכון הם רק ברמת מתחילים ולא יהיה זה הוגן (או מעשי) לדרוש מהם שיפגינו רמת מיומנות הנדרשת ממתכנת מקצועי. בנוסף לכך, חשוב לזכור שאצל מתחילים, גם משימות פשוטות מעוררות קשיים ובמקרים רבים, הם עומדים חסרי אונים מול שגיאות ואינם יודעים מה לעשות. במקרים אחרים, נוהגים התלמידים ל"גשש באפלה" (tinkering). כלומר, הם מנסים להטילא על התכנית

הואיל ושגיאות לוגיות נובעות בדרך-כלל מתכנון לקוי של האלגוריתם, הרי שניפויין דומה במידה מסוימת לתהליך של הוכחת נכונות התכנית. כאשר מנסים להוכיח נכונות תכנית, מתבססים על טענות הקלט ובעזרת מעבר על שלבי הביצוע, מנסים להוכיח את טענות הפלט. כאשר נתונה משימה והתכנית הקיימת אינה פותרת אותה, יש לחזור ולבדוק מהם הנתונים, מהם השלבים שמבצעת התכנית ומהן התוצאות שלה. הפלט של התכנית יכול לספק מדד על המצאותה של שגיאה לוגית. השגיאה הלוגית יכולה לנבוע כתוצאה מהנחות קלט שגויות, כתוצאה משלבי ביצוע חסרים, או ממיקום שגוי של שלבי ביצוע. יש חוקרים הטוענים שאם נקפיד בזמן תכנון הפתרון, להוכיח את נכונות האלגוריתם בכל שלב, הרי שכאשר נסיים לטפל בכל חלקי הפתרון ונתפור אותם יחדיו, מובטח שנקבל תכנית ללא שגיאות לוגיות.

### שגיאות ביצוע

מדובר בתכניות שכתובות על-פי חוקי תחביר תקינים, אולם בעת ההפעלה שלהן מתגלה בעייה. הבעייה יכולה לנבוע ממחסור בנתונים, מנתונים לא מתאימים שנוצרים תוך כדי הפעלת התכנית, או מקשרים לא מתאימים בין מרכיבי התכנית.

כל סוגי השגיאות יכולים לגרום לשגיאות ביצוע. יחד עם זאת, שגיאות ביצוע הן בעיקר תלויות-שפה, כיון שהוראה שתגרום בשפה אחת לשגיאת ביצוע, יכולה להיות חוקית בשפה אחרת. לדוגמה, ניתן לחשוב על שפה שבה הוצאת שורש ממספר שלילי לא תגרום לשגיאה ותחזיר את הערך המרוכב המתאים. כמו כן, יש חשיבות לקשר בין ההוראה ובין מצב המערכת ברגע נתון. כמעט כל הוראה יכולה להיתבצע באופן תקין במצב נתון אחד ולגרום לשגיאת ביצוע במצב אחר. לדוגמה, קריאת נתונים מקובץ תתבצע כהלכה כאשר יש בקובץ נתונים מתאימים, ותגרום לשגיאה כאשר הקובץ יהיה ריק.

שגיאות שמתגלות בזמן הביצוע "חוסכות" את שלב הגילוי של השגיאה ונותר רק לאתרה ולתקנה. כמובן

לתהליך של ניפוי שגיאות. בעקבות המחקר ובמקביל למודל שבנו, הם מסיקים שלצורך התהליך של ניפוי שגיאות, נדרשות המיומנויות הבאות: היכולת לחשב (evaluate) קוד באופן נכון, היכולת לאתר שגיאות על-ידי parsing של הקוד והתאמתו לתוצאות, היכולת ליצור קוד תקין כדי לתקן את השגיאה.

Littman בדק מתכנתים מקצועיים ומצא שהם השתמשו בשתי גישות בסיסיות להבנת תכניות: אסטרטגיה שיטתית (systematic strategy) ואסטרטגיה של "לפי הצורך" (as-needed strategy). הוא גם מצא קשר ישיר בין רמת הידע שרכשו המתכנתים בכל גישה ובין מידת ההצלחה שלהם לשכתב את התכנית. מחקרו מראה בברור שהגישה השיטתית עדיפה. מתכנתים שעבדו לפי האסטרטגיה הראשונה (השיטתית), עקבו אחר מעבר הנתונים והבקרה בתכנית כדי להבין את ההתנהגות הגלובלית שלה. מתכנתים שעבדו לפי האסטרטגיה השנייה, התמקדו בהתנהגות הלוקאלית. הגישה בה בוחר המתכנת, משפיעה על הידע שהוא רוכש לגבי התכנית ולפיכך משפיעה על מידת הצלחתו להבין את התכנית ולהיות מסוגל לתקנה או להכניס בה שינויים.

כדי להבין תכנית, צריך ללמוד על העצמים שלה, על הפעולות שהיא מבצעת ועל המרכיבים הפונקציונאליים שלה. אלו הם אספקטים סטטיים של התכנית (לא משתנים במהלך ההרצה) ולכן הוא מכנה את הידע של אספקטים אלו בשם ידע סטטי. כמו כן, כדי להבין את התכנית, יש צורך לדעת על הקשרים הסיבתיים בין מרכיבי התכנית - ידע סיבתי (knowledge causal).

במהלך קריאת התכנית והנסיון להבינה, יוצר המתכנת מודל מנטלי, המבוסס על הידע שהצליח לרכוש. במידה ורכש רק ידע סטטי על התכנית, הוא יוצר מודל מנטלי חלש. במידה והצליח לרכוש הן ידע סטטי והן ידע נסיבתי, הוא יוצר מודל מנטלי חזק. ככל שהמודל חזק יותר, יש סיכוי להצלחה גדולה יותר בשכתוב התכנית.

לפי Littman, המתכנתים שפעלו בגישת "לפי-הצורך" התמקדו בעובדות מקומיות ואילו המתכנתים שנקטו באסטרטגיה השיטתית עסקו בהבנת

תיקונים אקראיים, בתקווה שמזלם ישחק להם ואחד התיקונים או צרוף שלהם יתקן את השגיאה. הקושי בביצוע הניפוי ובעיקר חוסר המיומנות באסטרטגיות ניפוי יעילות, יוצרים "צואר בקבוק" גם בפיתוח מיומנויות תכנות אחרות. לפיכך, יש הטוענים שהוראת תכנות צריכה להתחיל מניתוח של מיומנויות (או אסטרטגיות) ניפוי.

### סקירת מחקרים

Gugerty ו-Olson חקרו ניפוי שגיאות אצל מתחילים ומתקדמים ואיבחנו שלוש אסטרטגיות עיקריות:

- גישת ההבנה (comprehension) - המתכנת מנסה להבין מה מבצעת התכנית, ומשווה למה שהתכנית אמורה לעשות. מתוך חוסר ההתאמות בין המצוי והרצוי, ניתן לאבחן את מוקדי הבעיות.
- גישה טופוגרפית - המתכנת נעזר ברמזים מתוך הפלט של התכנית, או במצבי התיכנות הפנימיים (בעזרת הדפסות ביניים), כדי לצמצם את איזור החיפוש של השגיאה.
- גישה סימפטומטית - המתכנת נעזר בידע קודם שיש לו על מצבים דומים. הוא מנסה להיזכר בסימפטומים המוכרים לו מהתנסויות קודמות, כדי לאבחן את מקור הבעיות. ברור ש"ספריית שגיאות" גדולה יותר יכולה לסייע במקרה זה ולפיכך, מומחים ישתמשו בגישה זו ביתר הצלחה.

Kessler ו-Anderson בדקו כיצד מתמודדים מתחילים עם התהליך של ניפוי שגיאות. הם גילו שרוב המתכנתים ניסו, בשלב ראשון, להבין מה עושה התכנית. לאחר-מכן, הריצו את התכנית עם נתונים "החשודים כיוצרי בעיות" וניסו ליצור מצבים שגויים. בשלב הבא, עסקו במציאת קטעי הקוד שגרמו לתוצאות השגויות, ולבסוף ניסו לתקן את השגיאה. הם מצאו שהשלב האחרון היה הקשה ביותר לביצוע, ללא קשר לצעדים שנקטו קודם לכן.

בעקבות המחקר, הם איבחנו ארבע אפיזודות בתהליך הניפוי: הבנת התכנית; חיפוש השגיאה; איתור מקום השגיאה; ולבסוף, תיקון השגיאה. הם מציעים מודל

הואיל וניתן לראות ניפוי שגיאות כמקרה פרטי של פתרון בעיות, מעניין לבחון את המחקרים שנערכו בהקשר הכללי יותר. לדוגמה, נזכיר את המחקרים שערך Schonfeld (מתימטיקאי, מרצה וחוקר בבית הספר לחינוך הנמצא במחלקה למתימטיקה של אוניברסיטת Berkely בקליפורניה, ארה"ב). Schonfeld התרשם מהרעיונות של Polya (בספר "כיצד פותרין" שהגרסה המקורית שלו נכתבה ב-1945, מציג Polya מודל בעל ארבעה שלבים לפתרון בעיות: הבנת הבעיה, תכנון הפתרון, ביצוע הפתרון, והערכה של הפתרון, כולל חיפוש של דרכים אחרות לפתור את אותה הבעיה) והתלהב מהמודל שלו, בעיקר כיוון שהוא עצמו השתמש במודל דומה כאשר עסק בפתרון בעיות מתימטיות. כאשר התברר לו שהמודל הזה אינו מסייע לסטודנטים באותה מידה שהיה ניתן לצפות ממנו, הוא הופתע והחליט לברר את המקור לאי ההתאמה הזאת. הוא הגדיר שתי שאלות מחקר מרכזיות: מהי הכוונה ב"חשיבה מתימטית"? וכיצד ניתן לסייע לתלמידים "לחשוב מתימטית"?

במסגרת זו, הוא ערך מחקר שכלל ניתוח פרוטוקולים של סטודנטים ומומחים שפתרו בעיות גיאומטריות שהוצגו להם. Schonfeld מצא שכדי להצליח בפתרון בעיות (בתחומי-דעת בעלי סמנטיקה עשירה), יש צורך בשני סוגים של החלטות: החלטות טקטיות (ביצועיות) והחלטות אסטרטגיות (ארגוניות). החלטות טקטיות כוללות נוהלים, אלגוריתמים והיוריסטיקות. ההיוריסטיקות מתייחסות להמלצות של Polya (צייר דיאגרמה, טפל במקרים מיוחדים) וגם לסוג ההיוריסטיקות שמשמשים בו באינטליגנציה מלאכותית. למשל, אם צריך לחשב שטח, ההחלטה האם להעזר בטריגונומטריה או האם להשתמש בגיאומטריה אנליטית, היא החלטה טקטית.

החלטות אסטרטגיות משפיעות על כיוון ההתקדמות של פתרון הבעיה ועל התחום ממנו נלקחים מקורות הידע שיש לפותר לצורך הפתרון. אלה הן החלטות שמסייעות לפותר לבחור בין מרחבים שונים של טקטיקות או שמביאות אותו לתקוף את הבעיה מכיוון אחר. הן מכונות גם החלטות

הקשרים בין החלקים השונים בד בבד עם הבנת המידע המקומי. מכאן ניתן להסיק שכדי להבין את התכנית טוב יותר, רצוי לרכוש לגביה הן ידע סטטי והן ידע נסיבתי. לשם כך, מומלץ לנקוט בגישה שיטתית שמקנה ידע נרחב יותר על ההתנהגות הגלובלית של התכנית.

McCoy-1 Carver רואים את תהליך הניפוי כמסתמך בעיקר על חיפוש "רמזים" בתכנית. רמזים שישפכו למתכנת אינדיקציה על מהות השגיאה ועל מיקומה. ללא רמזים אלו, המתכנת יאלץ לעבור על תכניתו באופן סדרתי, הוראה אחר הוראה, עד שיגיע למקור השגיאה. טענתם העיקרית היא, שתלמידים יוכלו ללמוד את מיומנויות הניפוי, אם מרכיביה יוצגו לפנייהם בצורה מפורשת וברורה. לפיכך, הם בנו מודל שמיועד לתפוס מיומנויות של ניפוי יעיל. הם הציגו לתלמידים את המודל על כל שלביו ולאחר מכן, עודדו אותם להשתמש בו לצורך הניפוי של תכניותיהם. המודל שלהם מתאר ניפוי יעיל כתהליך של חמישה שלבים ובהסתמך על המודל, הם בנו את "תכנית-הלימודים לניפוי שגיאות":

בדוק את התכנית. אם היא שגויה, בצע את הפעולות הבאות:

- א. שאל את עצמך: מהי הבעיה? איזה סוג של שגיאה היה יכול ליצור את הבעיה?
- ב. שאל את עצמך: האם התכנית מורכבת מתת-תכניות? היכן יכולה השגיאה להיות? (בתת-תכנית? בתוך מבנה בקרה? אחרי הוראה מסוימת?)
- ג. השתמש במידע שאספת עד כה, לצורך מציאת השגיאה. אחרת (אם לא הצלחת), קרא כל הוראה והחלט האם היא נכונה.
- ד. כאשר מצאת את השגיאה, שאל את עצמך: מה צריך להיות התיקון? בצע את התיקון.
- ה. בדוק את התכנית מחדש

בעקבות הניסוי של תכנית הלימודים שלהם, Carver ו-McCoy מצאו תוצאות מעודדות. כבר אחרי שיעור אחד שהתמקד במיומנויות של ניפוי, התלמידים התחילו לשאול "שאלות ניפוי" נכונות ולהעזר בספריית שגיאות שהועמדה לרשותם.



ואילו החלטות אסטרטגיות-ארגוניות יעסקו, לדוגמה, בשאלות הבאות:

- כאשר מגלים תוצאה שגויה, האם מדובר באלגוריתם שגוי או שמדובר במקרה גבולי? מהן ההשלכות של ההחלטה הזאת על הצעד הראשון בניפוי של התכנית?
- האם השגיאה נובעת מזרימה בקרה שגויה או שמדובר בבעייה עם המשתנים? מהן ההשלכות של ההחלטה הזאת על תהליך הניפוי?
- כאשר מגלים שבדיקת מרכיב מסוים מובילה למבוי סתום, האם לנסות לבודד קטע אחר? איזה?

ההחלטות הארגוניות נמצאות ברמה גבוהה יותר מההחלטות הביצועיות ואפשר להתייחס אליהן כאל אסטרטגיות-על. דוגמה נפוצה לאסטרטגיות-על כזו, היא הנסיון לבודד את השגיאה ולמצוא את המצב הפשוט ביותר שיוצר את אותה השגיאה. ברוב המקרים, השגיאה מתרחשת בסביבה "עשירה" ואם מצליחים לבודד אותה, הניפוי יכול להתבצע בקלות רבה יותר. באופן כללי, השליטה באסטרטגיות-על מאפיינת את המומחים בפתרון בעיות ומפרידה בינם לבין המתחילים. הרכישה של אסטרטגיות-על בניפוי שגיאות יכולה להתבצע בעקבות למידה על פתרון בעיות בכלל ועל אסטרטגיות של ניפוי בפרט.

### השלכות חינוכיות

התהליך של ניפוי שגיאות אינו קל ולרוב אינו יכול להילמד מעצמו. הוא כרוך ברמות שונות של הבנה ומיומנות ובא לידי ביטוי בכל שלבי הלימוד. כיצד ניתן ללמד את התלמידים לבצע ניפוי שגיאות וליישם את המסקנות של המחקרים שהוצגו? ראשית, חשוב לחזור ולציין את אורת הלימוד בכיתה ואת היחס של המורה כלפי שגיאות תלמידים מפתחים עמדות שליליות כלפי שגיאות ולכן גם קשה להם לקבל את עובדת "קיומן הבלתי נמנע" של שגיאות בתכנות. מורים שאינם מדגישים את הסובלנות לשגיאות, כחלק הכרחי מתהליך הלמידה, יגלו שתלמידיהם מתקשים לרכוש מיומנויות של ניפוי שגיאות. הם חוששים להודות

מנהליות או ארגוניות. החלטות מסוג זה הן חיוניות כאשר מתגלה מידע חדש או כאשר נשקל השימוש בטקטיקה שונה, ובמיוחד כאשר סדרה של כשלונות טקטיים מצביעה על כך שחייבים לעשות הערכת מצב ארגונית. בדוגמה שהוזכרה (חישוב שטח), ההחלטה להתעכב על חישוב השטח במשך 10 דקות (מתוך 20 דקות לפתרון הבעייה), היא החלטה ארגונית המתבצעת ללא תלות בשיטת החישוב של השטח.

כדי להמחיש את ההבדל בין שני סוגי ההחלטות, Schonfeld משתמש בדימוי מלחמתי. במסגרת זו, דוגמה להחלטה טקטית יכולה להיות באיזה כלי לוחמה להשתמש, ואילו ההחלטה האם לפתוח חזית לחימה נוספת היא החלטה ארגונית שעשויה לקבוע את תוצאות המלחמה לנצחון או מפלה כוללת. Schonfeld טוען שכדי להגיע למומחיות בפתרון בעיות, חייבים לשלוט בהחלטות הארגוניות דוקא. הוא גם טוען שחוסר ההצלחה בהוראת המתמטיקה מקורו בהזנחה ובהתעלמות מהתנהגויות ארגוניות (שהן מטה-קוגניטיביות, בדרך כלל). הואיל והחלטות טקטיות הן מקומיות ואינן נותנות תמונה כללית על התהליך של פתרון הבעייה, הרי שאין די בהן. כדי להיות מסוגל להתמודד עם בעייה בהצלחה ולהגיע למומחיות בפתרון בעיות, יש לשלוט גם בהחלטות אסטרטגיות-ארגוניות ולכן יש ללמד אותן במפורש.

אם ננסה ללכת בעקבות Schonfeld, הרי שניתן לזהות החלטות טקטיות ואסטרטגיות גם בתהליך של ניפוי שגיאות. החלטות טקטיות-ביצועיות יעסקו, לדוגמה, בשאלות הבאות:

- כאשר מחליטים לבודד קטע (או מרכיב) מסוים בתכנית, האם לבדוק אותו בעזרת ניפוי מלמטה כלפי מעלה או בעזרת הרצה "יבשה"?
- כאשר מחליטים לבדוק את המקרים הגבוליים, האם להתחיל מהמקרה הגבולי הפשוט ביותר, המורכב ביותר, הידוע ביותר?
- כאשר מחליטים לבדוק את זרימת הבקרה של התכנית, האם היא תיבדק בעזרת הרצה "יבשה" או בעזרת הדפסות ביקורת?

מקום השגיאה ותיקון השגיאה) וכן על ממצאיהם של Olson ו-Gugerty שזיהו את הקשר בין "ספרית שגיאות" גדולה יותר וההצלחה בניפוי שגיאות.

### הבנת תכניות תקינות

מחקרים רבים תומכים בחשיבות הרבה שיש להבנת התכנית כאחד השלבים הראשונים בניפוי שגיאות. לפיכך, כשלב ראשון, מומלץ לתת ללומדים קטעי תכנית (או תכניות מורכבות, לפי רמת התלמידים ומטרת התרגיל) ולבקשם להסביר מה התכנית עושה. בכל שלב שבו נלמד מבנה חדש, מומלץ לשלב בהוראה גם תרגיל של הבנת תכנית המכילה את המבנה החדש. בשלבים מסוימים, בהם מצטבר אוסף של מבנים או כלים, מומלץ לתרגל את התלמידים בהבנת תכנית המורכבת ממספר גורמים. הבנת תכניות תקינות כרוכה בביצוע משימות קוגניטיביות מורכבות ומוסכם על חוקרים רבים שהיא אינה מתבצעת באופן לינארי. הקורא צריך לזהות סכמות או תבניות שמרכיבות את התכנית, כמו סכמה של צובר, מונה, תפריט או מסגרת קלט.

Lukey מזהה מספר סוגים של תיאורים שיכולים לתרום בהבנת תכניות:

- חלוקת התכנית לקטעים (segmentation) - יישום העקרון של "הפרד ומשול". ניתן לבצע מעין חלוקה תחבירית של התכנית: תכנית מתחלקת לתת-תכניות, תת-תכנית מורכבת מהוראות שמורכבות מתוויות, משתנים, ביטויים ותנאים. הבעייה היא, שלעיתים, חלוקה כזו לא יוצרת קטעים שמתאימים לניתוח. לדוגמה, אם נחלק להוראות בודדות, לא נזהה סכמה של צובר.
- תיאור זרימת הנתונים: זיהוי הקלט והפלט של חלקי התכנית חיוני להבנתה. תיאור מסוג זה, חיוני במיוחד כאשר עוסקים בתכנות פונקציונלי.
- תיאור זרימת הבקרה של התכנית.
- תיאור ערכי המשתנים ותפקידם בתכנית.

בכך שעשו שגיאה ולפיכך קשה להם ללמוד משגיאות. לעומת זאת, אם המורה מקדיש זמן לטיפול בשגיאות ובניפוי, יש סיכוי שהתלמידים ירגישו חופשיים יותר להודות בשגיאות שלהם.

אפשר לתת לתלמידים רשימת אסטרטגיות ולהמליץ בפניהם להעזר ברשימה. אולם בכך לא די. ניפוי שגיאות דורש שליטה במיומנויות מורכבות של פתרון בעיות. הוא אינו נושא קשיח ולכן קשה ללמוד אותו בעזרת חוקים או מתכונים. אין ספק שיש חשיבות רבה לנסיון המצטבר של המתכנת. אך מה יעשו מתחילים שעדיין לא צברו "שעות תכנות" רבות? תלמידים אלו זקוקים להדרכה ברורה יותר.

Martin ו-Perkins מנתחים את הקשיים בניפוי שגיאות ומציעים להקל על התלמיד בעזרת שמירה על השימוש ה"חקרני" של השפה (נסה בעצמך, או בדוק במחשב) ועידוד השימוש באסטרטגיות בסיסיות של פתרון בעיות: מה כדאי לבדוק עכשיו? איך ניתן לתאר את הבעייה שנתקלת בה? מה יקרה אם תכניס את השינוי הבא? Clements ממליץ להביא את התלמידים לכך שיכלילו את מיומנויות הניפוי ככל האפשר. הוא מציע למורים לקדם אוריה של "למידה משגיאות" מעבר לדרישה של כתיבת תכניות נכונות, להדגיש ששגיאות הן מבורכות ולא ניתן למנוע אותן, להעזר בשאלות כמו מה עשינו כבר? מה צריך עדיין לעשות? מדוע אנו נוקטים בצעדים אלו? כיצד הם יעזרו לנו? מה יקרה כאשר ניתן את ההוראה הזאת? האם אתה יודע מדוע זה מופיע כאן? האם פתרנו את הבעייה שהתכווננו לפתור? מה בדיוק לא בסדר? האם הפתרון שלנו מתאים לבעייה הנתונה? האם הפתרון שלנו הגיוני?

מומלץ גם לתרגל את התלמידים בהדרגה. התירגול המדורג, בכל שלבי הלימוד, בא לענות על הבעייה החינוכית של תהליך הניפוי, אשר מחייב שימוש במיומנויות רבות ומורכבות, ולפיכך לא ניתן לטפל בו בשיטת "זבנג וגמרנו".

התרגול המדורג מסתמך בעיקר על ממצאיהם של Anderson ו-Kessler שזיהו ארבע אפיזודות במהלך תהליך הניפוי (הבנת התכנית, חיפוש השגיאה, איתור

בתחום-דעת מסוים אינו צריך "להתאמץ" בפתרון בעיות מאותו התחום, כיוון שהפתרון שלו הוא "מונחה סכמה" (schema driven) ועליו פשוט לשלוף מ"ספרית הסימפטומים" שלו את הפתרון המתאים לבעייה הנוכחית. בהסתמך על כל אלה, ברור שכדאי להשקיע בטיפוח של ספריות כאלה גם אצל התלמידים.

לסיכום, המאמר עסק בנושא של ניפוי שגיאות. נבדקה בו החשיבות של הוראתו לתלמידי תיכון, הוצגו בו חלק מהקשיים הדידקטיים והוצעו מספר שיטות להתמודד איתם. לדעתי, הנושא של ניפוי שגיאות חייב להיות מטופל בקורס תיכנות בכל שלביו. זהו נושא קשה להוראה ולפיכך יש צורך להקדיש לו זמן, התייחסות מכובדת, ולצייד את התלמידים במגוון של אסטרטגיות ואסטרטגיות-על שיסייעו להם בתהליך מורכב זה.

"מומחה (שחקן שח או מתימטיקאי) אינו צריך לנסות את כל הצעדים השגויים האפשריים כדי ללמוד מהי הדרך הנכונה. הוא 'מריח' את המסלולים המבטיחים ביותר ובוחר בהם מיד. מטרת ההוראה היא איפוא, במידה רבה, לחדד את 'חוש הריח' של התלמיד".

Hofstadter (1979)

### איתור שגיאות "ידועות מראש"

Anderson ו-Kessler מצאו שאחד השלבים החשובים בניפוי, הוא חיפוש השגיאה ואיתור מקומה המדויק. לאחר שהתלמידים יודעים כיצד לבצע הרצה יבשה של תכנית, נוכל לעבור לשלב מתקדם יותר של איתור שגיאות. הכוונה היא לתת לתלמידים תכנית המכילה במתכוון שגיאה מסוג מסוים. התלמידים יתבקשו למצוא את השגיאה ולתקנה. אפשר ואף רצוי לדרג את התרגילים: מהקל לקשה (תחילה שגיאות תחביריות ולאחר מכן שגיאות סמנטיות ולוגיות); מתכנית עם שגיאה אחת לתכנית עם מספר גדול של שגיאות; משגיאות הקשורות במבנה אחד או בהוראה אחת, לשגיאות הקשורות ביחס בין המבנים או המודולים השונים של התכנית; ניתן גם ל"שחק" עם וריאציות של שגיאות ולבקש מהתלמידים להמציא הוראות שגויות שיגרמו להופעתה של שגיאה מסוג מסוים.

### שינויים ותוספות לתכניות קבועות

בשלב הבא של התרגול המדורג, תינתן לתלמידים תכנית תקינה והם יתבקשו להכניס בה שינוי או להוסיף מרכיב חסר. תירגול כזה מאפשר להתמודד עם "פירוק לגורמים" של מרכיבי התכנית ומסייע להבין טוב יותר מהי תרומתו של כל מרכיב. כמו כן, הוא מאפשר למורה להתמקד בנקודות קושי ידועות מהספרות המחקרית ומנסיונו האישי.

### הרחבת "ספרית הסימפטומים"

Onorato ו-Schvaneveldt משווים בין מתחילים ומומחים בתחומים שונים. הם מצאו מונה משותף בין מומחים בתיכנות, פיסיקה, אלגברה ומשחק השחמט, לעומת מתחילים באותם התחומים. הם מציינים שבין יתר הגורמים, המומחיות מתבטאת בשימוש במידע כללי יותר, שימוש בידע ברמה גבוהה ובניצול מאגר התופעות המוכרות מהנסיון המצטבר. גם Olson ו-Gugerty עמדו על מקומו וחשיבותו של מאגר השגיאות (הגישה הסימפטומטית), וגם Schonfeld מצוין שמומחה

Iyengar (eds.), **Empirical Studies of Programmers**. Norwood: Ablex Pub Co., pp. 80-98.

- Onorato, L.A., Schvaneveldt, R.W. (1986). Programmer / Nonprogrammer Differences in Specifying Procedures to People and Computers. In Soloway, Iyengar (eds.), **Empirical Studies of Programmers**. Norwood: Ablex Pub., pp. 128-137.
- Papert, S. (1980). **Mindstorms: Children, Computers and Powerful ideas**. Basic Books Inc.
- Pattis, R.E. (1981). **Karel the Robot: A gentle introduction to the art of programming**. John Wiley & Sons.
- Perkins, D.N., Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. In Soloway, Iyengar (eds.), **Empirical Studies of Programmers**. Norwood: Ablex Pub., pp. 213-229.
- Perkins, D.N., Schwartz, S., Simmons, R. (1988). Instructional Strategies for the Problems of Novice Programmers. In Mayer (ed.), **Teaching and Learning Computer Programming, multiple research perspectives**. Lawrence Erlbaum Ass, pp. 153-178.
- Schoenfeld, A.H. (1983). Episodes and Executive Decisions in Mathematical Problem-Solving. In Lesh and Landaue (eds.), **Acquisition of mathematics concepts and processes**. Academic Press Inc.
- Spohrer, J.G., Soloway, E. (1986). Analyzing the high frequency bugs in novice programs. In Soloway, Iyengar (eds.), **Empirical Studies of Programmers**. Norwood: Ablex. pp. 230-251.

### ביבליוגרפיה

- Carver, S. McCoy (1988). Learning and Transfer of Debugging Skills: applying task analysis to curriculum design and assessment. In Mayer (ed), **Teaching and Learning Computer Programming, Multiple research perspectives**. Lawrence Erlbaum Associates, Inc., chapter 11.
- Clements, D.H., Guilo, D. (1984). Effects of computer programming on young children. In the **proceedings of Logo 84**.
- Clements, D.H. (1989). Representation and Monitoring. In **Logo Exchange**, May 89, pp. 28-30.
- Dijkstra, E.W. (1989). On the cruelty of really teaching computer science. In **Communications of the ACM**, Vol. 32, No. 12.
- Gugerty, L., Olson, G.M. (1986). Comprehension Differences in Debugging by Skilled and Novice Programmers. In Soloway, Iyengar (eds.), **Empirical Studies of Programmers**. Norwood: Ablex Pub.
- Harel, D. (1989). **The Science of Computing: Exploring the Nature and Power of Algorithms**. Addison-Wesley, Reading, MA.
- Hofstadter, D.R., (1979). **Godel, Escher, Bach: An eternal golden braid**. Vintage books.
- Kessler, C.M., Anderson, J.R. (1986). A model of novice debugging in LISP. In Soloway, Iyengar (eds.), **Empirical Studies of Programmers**. Norwood, NJ: Ablex Publishing Co., pp. 198-212.
- Littman, D.C., et al (1986). Mental Models and Software Maintenance. In Soloway,