

17, 16	<p>מערכת תוכנה, שלבי פיתוח אבטיפוס</p>	<p>אוסף כלים המאפשרים לייצר אוטו' את הקוד עפ"י אפיון התוכנה הדרושה ברמת-על (מחוללי יישומים). קיצור תהליך הפיתוח במערכות קטנות ובינוניות (למשל: מערכת ייצור דו"חות ממסד נתונים קיים). מקצר משלב התכן עד יצירת הקובץ. לא מקצר את זמן הניתוח, התכן והבדיקות, שמהווים יותר מ-2/3 מזמן הפיתוח. אחד השימושים הטובים ב-4GT הוא ליצירת אבטיפוס. פרדיגמת ה-4GT מתמקדת ביכולת להגדיר את התוכנה ברמה שקרובה לשפה הטבעית ובשימוש בסימון המקנה לתפקוד משמעות. היא כוללת: שפות לא פרוצדורליות לשאילתות על מסדי נתונים, יצירת דו"חות, התמרת נתונים, הגדרת עיצוב מסכים וקשרי הגומלין ביניהם, יצירת קודים, יכולת גרפית ברמה גבוהה ויכולת פרישת גיליון. כל אחד מהכלים קיים רק לגבי תחומי יישום מאוד מוגדרים. 4GT מתחילה עם איסוף הדרישות. במצב אידיאלי הלקוח יתאר את צרכיו והם יתורגמו באופן אוטומטי לאבטיפוס תפעולי. אולם המצב אינו בר ביצוע, שכן הלקוח עלול להיות לא בטוח לגבי מה שהוא צריך, דו-משמעי בהגדרת עובדות ידועות ועלול להיות לא מסוגל או לא לרצות להגדיר מידע במובן של ה-4GT יוכל לממש. בנוסף כלי ה-4GT אינם מתוחכמים מספיק על מנת להתאים לשפה טבעית באמת.</p>	<p>4GT 4th Generation Technique</p>
37,38, 39,42, 55,72, 77, 112	<p>מערכות תוכנה, תקנים לפיתוח</p>	<p>US-DOD-MIL-STD-498</p>	<p>498</p>
106	<p>מחלקות ועצמים, תיאור מילולי לא פורמלי</p>	<p>הציע גישה לקביעת מחלקות ועצמים בתיאור מילולי לא פורמלי.</p>	<p>Abbott</p>
122	<p>(תחזוקת תכנה)</p>	<p>בגרף בעמ' 122</p>	<p>Act</p>
97	<p>(יחסים בין עצמים תכן מונחה עצמים) Link</p>	<p>עצם המפעיל עצמים אחרים, אך הוא לעולם לא מופעל ע"י עצם אחר.</p>	<p>Actor</p>
120	<p>(סוג של תחזוקת תוכנה) תוכנה, תחזוקה</p>	<p>התאמת התוכנה לדורות חדשים של חומרה ותוכנת תשתית (למשל מע' הפעלה, db).</p>	<p>Adaptive Maintenance</p>
97	<p>(צורה של reuse מושגים מתקדמים במונחה עצמים) Reuse</p>	<p>למשל: ייצור מכוניות עפ"י אבטיפוס והוספת פריטים שונים לכל מכונית שהם, למעשה, רכיבים סטנדרטים Reusable הניתנים להרכבה בקומבינציות שונות, או שימוש בשגרה מתוך ספריה והתאמתה למטרה מסוימת.</p>	<p>Adjusting</p>
97	<p>(יחסים בין עצמים תכן מונחה עצמים) Link</p>	<p>עצם שיכול גם להפעיל עצמים אחרים וגם להיות מופעל ע"י עצמים אחרים. עצם נוצר בד"כ ע"י Actor או Agent אחר כדי לבצע פעולה מסוימת עבורו. לרוב, Agent נעלם אחרי שהוא מסיים את עבודתו.</p>	<p>Agent</p>
97, 99, 98	<p>(תכן מונחה עצמים) יחסים בין עצמים</p>	<p>צירוף המורכב מחלקיו הנקראים גם תכונות. הצירוף יכול להיות הכלה פיזית כגון מטוס המורכב ממנועים, כנפיים, מתקן נחיתה וכדו' או הכלה conceptual כגון בעל מניות ומניותיו.</p>	<p>Aggregate</p>
99	<p>(תכן מונחה עצמים) יחסים בין מחלקות</p>	<p>יחס של whole/part כאשר ההכלה היא פיזית או conceptual.</p>	<p>Aggregation</p>
31, 30		<p>בנו כלים אוטו' המבוססים על שורות קוד או FP ופרמטרים נוספים, כגון: מורכבות, חשיבות לארגון, איכות כוח-אדם.</p>	<p>Albrecht & Gaffney</p>
137, 139, 140	<p>שיטות מפרט פורמליות</p>	<p>מורכבים מ-4 חלקים: 1. מבוא המגדיר את סוג השם ואת הגדרות הקלט. 2. חלק תאור הפעולה בו הפעולה/ות מתוארות באופן לא פורמלי. 3. חלק מאפייני הפעולה בו מוגדר תחביר מבני הנתונים. 4. חלק האמיתה בו מוגדרת משמעות התנהגות מבני הנתונים. אמיתות אלו מתייחסות לפעולות המתבצעות כדי ליצור את הישות המבוקשת. דוגמאות לשימוש ב-Algebraic Specification – עמ' 140.</p>	<p>Algebraic Specification</p>
73, 39	<p>SSS</p>	<p>"קו בסיס מוקצה" לפרוייקט, נקבע ע"י הלקוח והמפתח לאחר אישור ה-, SRS, IRS, STP</p>	<p>Allocated Baseline</p>
119	<p>תוכנה, בדיקות</p>	<p>Factory Test. מתבצע באתר הפיתוח. הלקוח משתתף בו ובודק שהכל כמו שצריך. תרשים – עמ' 119. או: עשיית התוכנה in-house ועשיית tests בבית התוכנה עצמו.</p>	<p>Alpha Test</p>
96, 99	<p>(מושגים מתקדמים תכן מונחה עצמים) מודל העצמים מחלקות, יחסים בין</p>	<p>יחס המציין קשר משמעות בין 2 מחלקות. למשל: ורדים ונרות הם מחלקות בלתי תלויות אך שתיהן יכולות לשמש יחד לקישוט שולחן.</p>	<p>Association</p>
99	<p>(תכן מונחה עצמים) מחלקות, יחסים בין</p>	<p>יחס של משמעות. מתקיימים 3 אופנים של cardinality: 1. 1-to-1 2. 1-to-many 3. Many-to-many</p>	<p>Association</p>
98	<p>(סוגי סנכרון בין עצמים)</p>	<p>השולח עשוי לזוּם פעולה ללא התחשבות אם המקבל מצפה לקבל את ה-message.</p>	<p>Asynchronous</p>

	Synchronization		
8, 17, 35, 36	פרוייקטים המודל הספירלי Cocomo	סיווג פרוייקטים (עמ' 8), פיתח את המודל הספירלי (17), הציע 3 סוגי מודלים עבור COCOMO שפיתח (35,36).	B. Boehm
98	(סוגי סנכרון בין עצמים) Synchronous Synchronization	כמו synchronous רק שהשולח יותר על הפעולה אם המקבל אינו מוכן מיידית (למשל: א' מחייג לב' שיש לו שיחה ממתינה וא' מנתק).	Balking
119	תוכנה, בדיקות	בדיקת המע' מחדש באתר הלקוח בהשתתפות המפתח, באחריות הלקוח. ניתן לקבוע בחוזה מתי המע' עובר ללקוח (אחרי ה- α -test או אחרי ה- β -test). אם נקבע בחוזה שההתקנה של התוכנה היא באחריות המפתח אז גם ה- β -test הוא באחריות המפתח. תרשים – עמ' 119. <u>אז:</u> המפתח מוכן לתת את התוכנה ללקוחות מובחרים שיעשו על זה β -test ובתמורה יוכלו לרכוש את המע' אחרי שיתוקנו השגיאות במחיר מוזל. האחריות היא בידי המפתח.	Beta Test
96	מודל העצמים	הגדיר מושגים מתקדמים בתכן מונחה עצמים	Booch
44	פיתוח מערכת, אסטרטגיות	יחידה פונקציונלית: A cluster of modules that perform a specific software subfunction (Pressman). A version of software that meets a specified subset of the requirements that the completed software will meet (498).	Build
52	תוכנה ומסד נתונים	תרשים – עמ' 52.	Case Tools
121	(בקרת שינויים) CCCB, SCCB, PCCB	נהוג שקיימים ועדות בקרת שינויים (CCB) ברמות שונות ולכל רמה גבוהה יותר סמכויות אישור נרחבות יותר. הועדות מתכנסת אחת לפרק זמן קבוע ו/או ad-hoc וכל החלטותיהן נרשמות ומאושרות (בחתימות).	CCB <u>Change Control Board</u>
121	(בקרת שינויים) CCB	הרמה הגבוהה ביותר. בראשות מנהל בית התוכנה ובהשתתפות הלקוח. בסמכותה לאשר כל שינוי.	CCCB <u>Customer CCB</u>
39, 42, 112, 113	(סקר) SDD, CSC, PDR	בסוף התכן המפורט. סקר זה מתבצע על ה-SDD עבור כל CSC. בסקר זה, וסקר ה-PDR, המתבצעים בהשתתפות הנהלת המפתח ובד"כ גם בהשתתפות הלקוח, נסקרים מסמכי התכן באופן ביקורתי וניתן אישור להמשיך לתכן המפורט או לקידוד בהתאמה, אך תיתכן גם דרישה לשיפורים ולסקר חוזר. מטרת הסקר: לוודא שהתכן הלוגי נכון. זו הפעם האחרונה שמבצעים בקרה על ה-design. רשימת מה שנסקר – עמ' 113. ה-CDR צריך להיות מאושר ע"י הלקוח והוא מאפשר את (תחילת) הקידוד. בתום ה-CDR מקבלים נתח רציני של הכסף ולכן הוא קריטי.	CDR <u>Critical Design Review</u>
69	(תרשים זרימת נתונים) DFD	Control Flow Diagram – בניגוד ל-DFD מבנה זה כולל גם בקרה. דוגמה – עמ' 69.	CFD
121	בקרת תצורה CCB	כל מסמך שהושלם, כל תוכנה שנגמרה ועברה unit test (מבדק פנימי) וכל בדיקה פורמלית הם CI (Configuration Item) המחויבים בבקרת שינויים. כל CI מקבל זיהוי הכולל לפחות: 1. סוג ה-CI (מסמך, תוכנית, בדיקה). 2. שם ה-CI. 3. שם הפרוייקט/מוצר. 4. מס' גרסה/מהדורה. 5. תאריך ברגע ש-CI הוא תחת CM לא ניתן לבצע בו שינוי ללא קבלת הרשאה מהגורם המוסמך – ועדת בקרת השינויים – CCB (Change Control Board). כאשר שינוי אושר ובוצע גרסת ה-CI משתנה והגרסה הקודמת נשמרת. יש חשיבות לשמירת גרסאות קודמות. למשל, כאשר יש תוכנות שבגרסה מסוימת עובדות זו עם זו, אבל לאחר כמה גרסאות יש בעיה – ניתן לשחזר את השינויים ולדעת איזו גרסה גורמת בעיה.	CI
71, 104	Object Oriented Methods	פיתחו שיטה קלאסית לסיווג מחלקות ועצמים - Structure - יחסי "is-a" ו-"part-of"	Coad & Yourdon
20	מערכות מורשה	Other system - מע' חיצונית איתן היישום פועל שפת תכנות המתאימה לעיבוד נתונים עסקיים.	COBOL
30, 32, 33, 34, 35, 36	Boehm פרוייקטים, סוגים כופלי מאמץ מודל יישומי רישום מוקדם המודל הפוסט ארכיטקטוני	Constructive COst MOdel. מודל שמבוסס על שורות קוד. Boehm הציע 3 סוגי מודלים עבור COCOMO1: 1. מודל בסיסי שמתחשב רק במס' ש"ק ולא מתחשב ב-cost drivers. 2. מודל ביניים. 3. מודל מורחב בו ה-cost drivers נמדדים בנפרד עבור כל שלב בפרוייקט. מודל ה-COCOMO שיקף את הנהלים היומיים של תכנון תוכנה. עקב שימוש חוזר של תוכנות קיימות, בניית מערכות חדשות ע"י שימוש ברכיבים של תוכנות מדף והשקעת מאמץ מוגבר בתכנון וניהול של תהליך פיתוח התוכנה נגרמה	COCOMO

		<p>בעייתיות ביישום מודל COCOMO1 והיה צריך להמציא מחדש את המודל ולהתאימו לשנות ה-90. ב-COCOMO2 אין סיווג הפרוייקט והוא מאפשר להעריך את העלות, המאמצים ולוח הזמנים כאשר מתכנים פיתוח של תוכנה חדשה. הוא כולל מודלים חלופיים שכל אחד מהם מאפשר יותר דיוק ככל שמתקדמים בתכנון הפרוייקט ובתהליך התכנון:</p> <ol style="list-style-type: none"> 1. המודל היישומי. 2. רישום מוקדם. 3. המודל הפוסט ארכיטקטוני. 	
37	(תקנים לפיתוח מע' תוכנה - MIL)	מדריך להפעלת המחשב והציוד – המדריך לא תלוי בפרוייקט אלא בציוד שמשתמשים בפרוייקט.	COM Computer Operation Manual
100	(תפקידי מחלקות ועצמים בניית ותכן) מחלקות ועצמים	המידה בה מחלקה או עצם כולל את כל המאפיינים והמשמעויות של ההפשטה.	Completeness
32	FP Feature Point Effort Person Month	מקדם סיבוכיות לפרוייקט כולו. למשל: אם זה פרוייקט שכבר עשינו הרבה כאלו ← מקדם נמוך. אם זה פרוייקט חדש ← מקדם גבוה. מופיע בתרשים בעמ' 32.	Complexity Multiplier
120		CM - ראה בקרת תצורה	Configuration Management
97	(צורה של reuse מושגים מתקדמים במונחה עצמים) Reuse	למשל: ייצור אלפי מכוניות מאותו סוג עפ"י אבטיפוס, או העתקת קטע קוד לתוכניות שונות.	Copying / Cloning
120	(סוג של תחזוקת תוכנה) תוכנה, תחזוקה	איתור ותיקון תקלות.	Corrective Maintenance
37		המדריך למתכנת המתחזק. מופיע בתרשים עמ' 37.	CPM Computer Programming Manual
38 113		Computer Software Components. המודלים הראשיים של המע'.	CSC
42, 38 112		Software Development. רכיבי מע' תוכנה. Computer Software Configuration Item (תוכנה). נמצא במסמך SSS.	CSCI
62, 64-69 84	(שיטות ניתוח) המודל הפונקציונלי מתודולוגיות הכנת מפרט דרישות תוכנה CFD Data Structure Oriented Object Oriented שיטות מפרט פורמליות	<p>השיטה מבוססת על התפיסה שמידע נכנס למע', עובר בה התמרה (transform) ויוצא. מקובל לתאר את זרימת המידע דרך המע' באמצעות DFD עם מילון נתונים ותיאור מילולי של כל פונקציה.</p> <p>תרשים – עמ' 64. סימוני DFD – עמ' 64.</p> <p>הניתוח באמצעות DFD מתבצע עפ"י רמות פירוט כאשר רמה 0 היא תיאור כל המע' כהליך (בועה) אחד אליו נכנסים הקלטים וממנו יוצאים הפלטים. כל רמה נוספת היא פירוט נוסף של התהליכים עד שמגיעים לרמת פירוט אותה ניתן לתאר במדויק ע"י אלגוריתמים, באופן מילולי ו/או בנוסחאות.</p> <p>דוגמאות – עמ' 65-68.</p> <p>ה-DFD אינו תרשים זרימה כי אין הוא כולל את הבקרה, כגון: התניות ולולאות. כל חץ של DFD מייצג פרטי מידע והתוכן של מידע זה מפורט במדויק במילון הנתונים.</p>	<p>Data Flow Oriented</p> <p>שיטות ניתוח מונחות זרימת נתונים</p> <p>DFD</p>
79-82	(שיטות תכן) ארכיטקטורת תכן Transform Analysis Transaction Analysis	<p>משתמש במאפייני זרימת המידע כדי להגיע למבנה התוכנית/התוכנה.</p> <p>כבסיס משמש ה-DFD שב-SRS ממנו נגזר השימוש ב-Transform Analysis או Transaction Analysis עבור התכן.</p> <p>צעדי התכן:</p> <p>צעדים 1-3 משותפים ל-transform ול-transaction:</p> <ol style="list-style-type: none"> 1. סקור מחדש את המודל הבסיסי של המע' (רמת על של זרימת מידע, כמתואר ברמה ה-1 וה-2 של ה-DFD): מסתכלים במסמך SRS, מוצאים את ה-DFD יש בו 2 רמות מעבר לרמה 0. יכול להיות שהאנשים שעשו את התכן אינם אותם אנשים שעושים את הניתוח. מה גם שיכול להיות שעבר הרבה זמן מאז שנעשה ה-SRS. 2. עדן את ה-DFD (תאר רמה 3 ויותר של ה-DFD – עוד פירוקים לבועות). 3. קבע האם ל-DFD יש מאפייני transform ו-transaction. <p>עבור ה-transform:</p> <ol style="list-style-type: none"> 4. בודד את מרכזי ה-transform ע"י קביעת גבולות נתונים נכנסים ונתונים יוצאים (שרטוט החלוקה על ה-DFD: איזה בועות מבצעות פלט ראשי/משני ואיזה קלט ראשי/משני). 5. בצע פירוק לגורמים ראשוני – קביעת מבנה התוכנה במודלים ברמות העל שמבצעים החלטות ומודולים ברמות הנמוכות שמבצעים את עבודות הקלט, העיבוד והפלט (מבנה מדרגי). 6. בצע פירוק לגורמים מסדר שני – מפה כל בועה ב-DFD למודול המתאים 	<p>Data Flow Oriented Design</p> <p>שיטות תכן מונחות זרימת נתונים</p>

		<p>במבנה התוכנה. התחל ממרכז ה-transform והמשך בתנועה לאורך המסלולים הנכנסים ואח"כ לאורך המסלולים היוצאים מהמרכז.</p> <p>7. עדן את מבנה התוכנה. צרף או הפרד מודולים כדי לקבל פירוק טוב עם ליכוד טוב וצימוד מינימלי שניתן לממשו, לבדקו ולתחזקו ללא קושי, בלבול ובעיות.</p> <p>עבור transaction:</p> <p>4. זהה מרכזי transaction ומאפייני הזרימה בכל מסלול פעולה.</p> <p>5. מפה את ה-DFD למבנה תוכנה מתאים לביצוע ה-transaction (בד"כ כניסה אחת ומס' רב של יציאות מופעלות באופן מותנה).</p> <p>6. פרק ועדן את מבנה ה-transaction ואת המבנים של כל מסלולי הפעולה.</p> <p>7. עדן את מבנה התוכנה (כמו לגבי transform).</p> <p>היוריסטיקות תכן:</p> <p>1. חן את מבנה התוכנה הראשי כדי לצמצם צימוד ולשפר ליכוד (כמה שיותר עצמאות ואי-תלות פונקציונלית).</p> <p>2. צמצם מבנים עם fan-out נרחבת. שאף ל-fan-in ככל שהרמות עולות. דוגמה – עמ' 81-82.</p> <p>3. שמור את טווח ההשפעה של מודול במסגרת טווח הבקרה של מודול זה.</p> <p>4. בחן את המנשקים בין המודולים כדי לצמצם מורכבות ויתירות ולשפר עקביות.</p> <p>5. הגדר מודולים עם תפקוד הניתן לצפייה מראש, אך הימנע ממודולים בעלי מגבלות יתר (למשל: מודול שמטפל במדפסות X ומודול שמטפל במדפסות Y, כאשר X ו-Y 2 מדפסות מאוד דומות).</p> <p>6. שאף למודולים בעלי נקודת כניסה יחידה ונקודת יציאה יחידה. הימנע ממודולים בעלי קשרים פתולוגיים (דהיינו כניסות לאמצע מודול ו/או יציאות מאמצע המודול).</p> <p>7. ארוז את התוכנה בהתבסס על אילוצי התכן ודרישות ניידות.</p>	
70, 69	<p>(שיטות ניתוח) <</p> <p>מתודולוגית הכנת מפרט דרישות תוכנה <</p> <p>DFD <</p> <p>Object Oriented <</p> <p>שיטות מפרט פורמליות <</p> <p>Warnier <</p> <p>JSD <</p>	<p>שיטה זו מייצגת מידע באופן מדרגי באמצעות תרשימי Warnier ואת תוכן היישום על"י מתן מענה ל-3 השאלות:</p> <p>1. מהם פריטי המידע שצריך לעבד.</p> <p>2. מי/מה הם יצרני וצרכני המידע.</p> <p>3. כיצד כל יצרן/צרכן רואה את המידע ביחס לאחרים.</p> <p>בשיטה זו משתמשים בתרשים ישויות שדומה ל-DFD, בו בועות מייצגות צרכני/יצרני מידע וחצים ביניהם את המידע העובר ביניהם.</p> <p>דוגמה – עמ' 70.</p>	<p>Data Structure Oriented</p> <p>שיטות ניתוח מונחות מבנה נתונים</p>
37, 77, 43	<p>(תקנים (MIL) לפיתוח מע' תוכנה)</p>	<p>תיאור מבנה מסד נתונים שהוא מסובך/מורכב (אופציונלי: אם אין מסד נתונים מורכב לא צריך). שייך לתיק העיצוב בנוהל מפת"ח.</p>	<p>DBDD</p> <p>Database Design Description</p>
72	<p>(חלק מ-STP)</p>	<p>תוכניות מבחן. עושים מבחן על CSCI מסוים שמקבל קלט ושולח פלט ל-CSCI אחר. אם לא תכנתנו את ה-CSCI ששולח את הקלט אזי ה-driver הוא תוכנית שמדמה את הקלט.</p>	<p>Drivers</p>
30	<p>(הערכת עלות תוכנה)</p>	<p>Deliverable Source Instructions - הערכה לפי מס' שורות קוד</p>	<p>DSI</p>
30	<p>(הערכת עלות תוכנה)</p>	<p>Delivered Source Lines - הערכה לפי מס' שורות קוד</p>	<p>DSL</p>
129	<p>(מדידת איכות תוכנה) <</p> <p>איכות תוכנה, מטריקות <</p>	<p>מדד לפרוייקט. פותח על"י חיל האוויר האמריקני. ה-DSQI מקבל ערכים בין 0 ל-1. את ה-DSQI של הפרוייקט בפיתוח משווים ל-DSQI של פרוייקטים דומים שהסתיימו. נוסחה – עמ' 129.</p>	<p>DSQI</p> <p>Design Structure Quality Index</p>
69, 83, 84	<p>(שיטות ניתוח ותכן) <</p> <p>תכן מונחה, מבנה נתונים <</p> <p>JSD <</p> <p>LCP <</p> <p>Warnier <</p>	<p>Data Structured System Development. תרשים ישויות – עמ' 69.</p> <p>התכן הלוגי ב-DSSD מתמקד בפלטים, במנשקים ובתכן תהליכי התוכנה. התכן הפיזי נובע מהתכן הלוגי ומתמקד באריזת התוכנה כדי להשיג את הביצועים הנדרשים ושאר אילוצי התכן. התכן מתועד באמצעות תרשימי Warnier. תרשים – עמ' 84.</p>	<p>DSSD</p>
123	<p>(הבטחת איכות תוכנה)</p>	<p>הגדירו הבטחת איכות תוכנה</p>	<p>Dunn & Ullman</p>
121	<p>(בקרת שינויים) <</p> <p>PTR <</p> <p>בקרת תצורה <</p>	<p>Engineering Change Proposal - בקשות לשינויים מתקבלות כ-ECP לכל שינוי שנדרש שלא נובע מתקלה/טעות.</p>	<p>ECP</p>
32	<p>FP <</p>	<p>המאמץ לפיתוח הפרוייקט נמדד בחודשי אדם. נוסחה לחישוב עמ' 32.</p>	<p>Effort</p>
32	<p>(סוגי פרוייקטים ב-COCOMO1) <</p> <p>פרוייקטים, סיווג <</p>	<p>1. אוניוקה – תעופה: המערכות המוטסות עצמן (בתוך המטוס).</p> <p>2. Command & Control מתוחכם (למשל: מע' בקרה אווירית).</p> <p>3. עיבוד תנועות מורכב.</p> <p>4. Real Time.</p> <p>מכל גודל שהוא (גם מע' קטנה שהיא מורכבת ומסובכת).</p>	<p>Embedded (משובצות)</p>
63, 62	<p>(שיטות ניתוח - בניית מודל ניתוח) <</p> <p>מודל הנתונים <</p>	<p>Entity Relationship Diagram – מודל הנתונים מתוך בניית מודל ניתוח. מתאר את הנתונים והיחסים ביניהם. דוגמה בעמ' 63.</p>	<p>ERD</p>
44, 47, 45	<p>פיתוח מערכת, אסטרטגיות <</p> <p>Build <</p>	<p>מניחים שצורכי המשתמש אינם מובנים היטב והדרישות לא ניתנות להגדרה בתחילת ביצוע הפרוייקט, לכן מבצעים את שני השלבים הראשונים:</p> <p>1. קביעת צרכי המשתמש.</p>	<p>Evolutionary</p>

	<ul style="list-style-type: none"> המודל הספירלי Grand Design Incremental Reengineering 	<p>2. הגדרת דרישות התוכנה. באופן חלקי ועפי" זה מיישמים חלקית את המערכת (או אבטיפוס שלה), אח"כ חוזרים לשלב הראשון ומעדנים אותו, מיישמים build נוסף, וחוזרים שוב על כל השלבים. דומה למודל הספירלי. 3. תכן המערכת. 4. יישום המערכת – קידוד. 5. מבחני קבלה, תיקונים ומסירה – המבחנים על המע'. טבלת השוואה בין הגישות – עמ' 45. טבלת סיכונים והערפות – עמ' 47.</p>	<p>פיתוח התפתחותי</p>
		<p>ראה Data Flow Oriented Design.</p>	<p>Fan In/Out</p>
32	<ul style="list-style-type: none"> FP Complexity Multiplier Effort 	<p>נוסחה לחישוב – עמ' 32.</p>	<p>Feature Point</p>
20	<ul style="list-style-type: none"> מערכות מורשה 	<p>שפה המתאימה לתכנות מדעי או מתמטי. קיימת תמיכה מאוד מוגבלת לבניית מסדי נתונים.</p>	<p>FORTRAN</p>
30, 32, 31	<ul style="list-style-type: none"> (חישוב עלות פרויקט) Effort Feature Point Complexity Multiplier Person Month 	<p>Function Points. למשל: מס' הקלטים ממשמש, מס' הפלטים למשתמש, מס' השאלות של המשתמש, מס' הקבצים, מס' המנשקים החיצוניים וכדו'. גישה זו מציגה כמותית מה שתוכנית יכולה לעשות. הערכות אלו מתאימות למערכות מנהליות (ענ"א), אך לא למע' בקרה, מע' זמן-אמת או מע' משובצות מחשב. ניתן נקודות התפקוד עוזר למפתחים ומשתמשים להעריך את גודל מורכבות יישומי התוכנה במידה שזה יעיל למשתמשי התוכנה. נקודות התפקוד אינן מטריצות מושלמות אבל הן שימושיות. הן עצמאיות, טכניות, עקביות, חוזרות על עצמן ועוזרות לתקנן נתונים. הן ניתנות להשוואה ונותנות מסגרת לציפיות הלקוח. הן מתמקדות בתפקודיות שהמשתמש צריך, כפי שמופנה ע"י התוכנה, בכך שמשתרשים מדידה בלתי ישירה של דרישות התוכנה. למשל: יישום ספר חשבונות שמכיל 1,500 FP יישאר בגודל זהה ב-COBOL, ++C ו-SmallTalk. המטרה של FP היא לבדוק גודל ולא מאמץ ולכן אין ל-FP תלות טכנולוגית עקב המאמץ הדרוש ביישום טכנולוגיות שונות. טבלה לחישוב Effort - עמ' 32. לכל FP יש משקל. למי שיש חשיבות יותר גדולה מקבל משקל יותר גבוה.</p>	<p>FP</p>
39	<ul style="list-style-type: none"> (סקר) 	<p>בסופו יש Product Baseline. ה-FQR הוא הסקר האחרון שנערך בתום המע' ובוה מסיימים את המע' מבחינת 498.</p>	<p>FQR Formal Qualification Review</p>
97	<ul style="list-style-type: none"> (תכן מונחה עצמים) 	<p>Friend של מחלקה היא מחלקה שמותר לה לגשת לחלק ה-private שלה ולכן כמובן גם לחלק ה-protected שלה.</p>	<p>Friend</p>
37	<ul style="list-style-type: none"> (תקנים (MIL) לפיתוח מע' תוכנה) 	<p>מדריך לסיוע בקושחה: איך יצרו את הקושחה, איך מטפלים בה, פרמטרים וכו'.</p>	<p>FSM Firmware Support Manual</p>
55, 39	<ul style="list-style-type: none"> SSS 	<p>קו הבסיס הראשון שקובע את פונקציונליות המערכת. בתום ה- SDR הלקוח מאשר את ה- SSS והוא קו הבסיס הפונקציונלי. אם רוצים לסטות ממנו זה כרוך בעלויות, אישורים של דרגים בכירים ושינויים בל"ז. מאפשר לראות מה ניתן לבצע במקביל.</p>	<p>Functional Baseline</p>
35			<p>Gantt Chart</p>
99	<ul style="list-style-type: none"> (תכן מונחה עצמים) מחלקות, יחסים בין 	<p>דהיינו, יחס "Is-A", למשל: המחלקה "ורדים" היא סוג (is a) של המחלקה "פרחים".</p>	<p>Generalization / Specialization</p>
45, 44	<ul style="list-style-type: none"> פיתוח מערכת, אסטרטגיות מפל המים, מודל Increment Evolutionary Reengineering 	<p>מבצעים פעם אחת את השלבים: 1. קביעת צרכי המשתמש. 2. הגדרת דרישות התוכנה. 3. תכן המערכת. 4. יישום המערכת – קידוד. 5. מבחני קבלה, תיקונים ומסירה – המבחנים על המע'. דומה למודל מפל המים שכן מבצעים הכל פעם אחת, וחוזרים על שלבים רק אם אין ברירה. טבלת השוואה בין הגישות – עמ' 45.</p>	<p>פיתוח כוללני</p>
98	<ul style="list-style-type: none"> (סוג סנכרון) Synchronization 	<p>קיימים מס' תהליכים פעילים וה-clients משתפים פעולה עמ"נ להשיג מניעה הדדית.</p>	<p>Guarded</p>
128	<ul style="list-style-type: none"> (מדידת איכות תוכנה) איכות תוכנה, מטריקות 	<p>השימוש בממד זה בקושי קיים. הממד הוא אובייקטיבי, אבסולוטי. נוסחה – עמ' 128.</p>	<p>Helstead's Complexity Measure</p>
37, 38, 43, 42	<ul style="list-style-type: none"> (תקנים (MIL) לפיתוח מע' תוכנה) 	<p>Hardware Required Analysis החומרה שהתוכנה רצה עליה (המחשב, הציוד ההיקפי, הציוד הנלווה) ← טוב כדי להתחיל לבצע את שלבי הרכישה לפני סיום הפרוייקט.</p>	<p>HRS</p>
38	<ul style="list-style-type: none"> (תקנים (MIL) לפיתוח מע' תוכנה) 	<p>Hardware Software Configuration Item (חומרה, קושחה, אונשה). נמצא במסמך SSS.</p>	<p>HWCI</p>

	SSS <		
,37 77,43	(תקנים (MIL לפיתוח מע' תוכנה)	תיאור מבנה המנשקים: אם יש מנשקים מסובכים וה- IRS אינו ברמת פירוט מספקת חייבים לכתוב IDD עבור אותם מנשקים. שייך לתיק העיצוב בנוהל מפת"ח.	IDD <u>Interface Design Description</u>
7 130	הנדסת תוכנה <	נותנים הגדרה להנדסת תוכנה בעמ' 7. מציעים אינדקס לבשלות תוכנה (SMI) בעמ' 130	IEEE
,19 45,44	פיתוח מערכת, אסטרטגיות Build < Grand Design < Evolutionary < Reengineering <	מבצעים פעם אחת את: 1. קביעת צרכי המשתמש. 2. הגדרת דרישות התוכנה. ואח"כ מבצעים את יתר השלבים, כל פעם עבור build אחד למימוש יכולת אחת: 3. תכן המערכת. 4. יישום המערכת – קידוד. 5. מבחני קבלה, תיקונים ומסירה – המבחנים על המע'. טבלת השוואה בין הגישות – עמ' 45.	Incremental <u>פיתוח מדורג</u>
118	בדיקות חיצוניות < בדיקות פנימיות <	רצף בדיקות המתואר בשרטוט בעמ' 118	Incremental Testing
94,99	(תכן מונחה עצמים) מחלקות, יחסים בין <	יחס בו מחלקה אחת יורשת את המבנה או את ההתנהגות של מחלקה אחרת או של מספר מחלקות אחרות. המחלקה המורשת היא superclass והמחלקה היורשת היא subclass. single \ multiple inheritance - ירושה ממחלקה אחת / מס' מחלקות.	Inheritance
,116 117	(בדיקות תוכנה) בדיקות פנימיות <	בודקים מוצר מול תקן. איך התוכנית עושה. דוגמה: Code Review: בדיקת הקוד עצמו (בדיקת מקרי קצה, ערכים התחלתיים וכיו"ב. רשימה – עמ' 116-117.	Inspections
100	(תכן מונחה עצמים) מחלקות, יחסים בין <	יחס של מחלקה מ-type מסוים המאפשר ל-subclass שלו להיות מ-type אחר אשר נקבע ע"י instantiation. למשל: מחלקה מסוג תור FIFO יכולה להתממש כתור של שלמים, כתור של שמות וכתור של display items. מחלקה כזאת נקראת parameterized או generic וללא instantiation אין לו instances (subclass או object) – בדומה ל-template.	Instantiation
,37,38 ,42 72,43		מפרט דרישות מנשקים. יחד עם SRS הם מהווים המקבילה לתיק האפיון בנוהל מפת"ח (Software Requirements Analysis). יש IRS כמספר המנשקים שיש לתאר (מנשקים בין CSCI לבין עצמם ומנשקים בין HWCI לבין עצמם).	IRS <u>Interface Requirements Specification</u>
94 104	(תכן מונחה עצמים) הורשה/מדרג < Coad and Yourdon <	מופיע גם בשיטה הקלאסית לקביעת מחלקות ועצמים לפי Coad and Yourdon. צורת הכללה.	Is-A
70,71 83	(שיטה לניתוח ותכן) תכן מונחה מבנה נתונים < LCP < DSSD < Data Structure Oriented <	שיטת Jackson System Development: דוגמה לשיטת data structure oriented. שיטה זו מורכבת מ-6 צעדים. 3 הצעדים הראשונים קשורים לניתוח התוכנה (אפיון) ו-3 האחרים לתכן: 1. Entity Action - צעד "ישויות פעילויות": בשלב זה מזהים את הישויות – יצרני וצרכני המידע (או המע') ואת הפעילויות – האירועים שקורים בעולם האמיתי והמשפיעים על הישויות. זאת מתוך תיאור מילולי של המע'. 2. Entity Structure - צעד מבנה ישות: בשלב זה מסדרים את הפעילויות המשפיעות על הישויות (שזוהו בצעד ה-1) עפ"י זמנים שמיצגים אותם ע"י תרשים Jackson. דוגמה – עמ' 70. 3. Initial Mode - צעד מודל ראשוני: בשלב זה בונים מפרט של המע' כמודל של העולם האמיתי שתואר בצעדים 1 ו-2. דוגמה – עמ' 71. 4. Function. 5. System Timing. 6. Implementation. בתכן הדגש הוא על 3 השלבים האחרונים.	JSD
,100 106	(שיטה לקביעת מחלקות ועצמים בתכן מונחה עצמים) מחלקות ועצמים < Mechanism <	מחלקה או עצם המהווה מונח במרחב הבעיה. זיהוי key abstraction הוא בעיקר עפ"י מומחי התחום. למשל: משתמש בכספומט מדבר על משיכות, בירור יתרה וכו'. ייתכן שהמפתח יצטרך להוסיף key abstraction כגון מנהל מסכים, מסד נתונים וכו'. הדרך הטובה ביותר לזיהוי key abstraction היא בהסתמך על מחלקות ועצמים קיימים. אם אין reusable abstractions מומלץ להשתמש בתרחישים לזיהויים. עידון ה-key abstraction: לאחר שזוהו ה-key abstractions צריך להעריכם עפ"י שיקולי Primitiveness, Completeness, Sufficiency, Cohesion, Coupling. כתוצאה מכך ייתכן ויווצרו abstractions חדשים לעתים ע"י איחוד/פיצול/העלאה ברמה (מעצם למחלקה) או הורדה ברמה. בפרט אם אין תשובות ברורות לשאלות: כיצד יוצרו עצמים ממחלקה זו? האם ניתן להעתיק או להשמיד עצם ממחלקה זו? אלו פעולות ניתן לבצע על עצם זה? אזי כדאי מעוד לבצע עידון או עדכון. ראה שיטות מפרט פורמליות.	Key Abstraction
			Language-based Formal

			Specification
95	(עקרונות מודל העצמים)		ה-type של כל המשתנים והביטויים נקבעים בזמן ההרצה (runtime).
83	(שיטה לתכן) < תכן מונחה מבנה < נתונים < JSD < DSSD < Warnier		בניה לוגית של תוכניות - Logical Construction of Programs. ב-LCP משתמשים בתרשימי Warnier לייצוג התכן. הן מבני הנתונים והן התהליכים מתוארים בצורה מדרגית באמצעות תרשימי Warnier. במבנים מורכבים משתמשים גם באלגברה בוליאנית.
99, 97	(תכן מונחה עצמים) < יחסים בין עצמים < Aggregation < Actor < Server < Agent < היראות < Synchronization		קישור פיזי או מושגי/תפיסתי בין עצמים. קו המקשר בין שני ייצוגי עצמים מייצג Link ביניהם ומשמעותו שה-messages יכולים לעבור דרכו. ה-Link מציין קישור ספציפי באמצעותו עצם אחד, ה-client, מפעיל את השירותים של עצם אחר, ה-supplier. העצמים המקושרים ב-link יכולים להיות בעלי אחד התפקידים הבאים: 1. Actor. 2. Server. 3. Agent
30	(הערכת עלות תוכנה)		Lines of Code (שורות קוד – ש"ק).
128	(מדידת איכות תוכנה) < איכות תוכנה, < מטריקות		מדידת המורכבות של תוכנית כמבוסס על גרף התוכנית. נוסחה – עמ' 128.
125	(מדדי איכות תוכנה)		הגדיר מדדי איכות תוכנה
107	(שיטות לקביעת מחלקות ועצמים בתכן מונחה עצמים) < מחלקות ועצמים < Key Abstraction		מבנה על-פיו עצמים משתפים פעולה לספק התנהגות מסוימת עמ"נ לענות על דרישת הבעיה. Mechanism מייצג דגם של התנהגות. קביעת mechanism היא החלטת תכן אסטרטגית בנוגע לשיתוף הפעולה של מס' רב של עצמים מסוגים שונים.
99, 96	מודל העצמים < היראות < Synchronization < Link		הפעולה שעצם אחד מבצע על עצם אחר.
100	(תכן מונחה עצמים) < מחלקות, יחסים בין		מוגדר כמחלקה של מחלקה, דהיינו מחלקה אשר ה-instances שלה הם מחלקות. הנתונים וההתנהגות שהם class-specific ממוקמים ב-metaclass של מחלקה זו. Metaclasses מאפשרים התמחות של ההתנהגות של המחלקה ובלעדיהם כל המחלקות היו מתנהגות באותו אופן. למשל: ה-metaclass של חשבונות מייצר מחלקות של חשבונות, דהיינו מגדיר אותם (class שמגדיר class).
34, 33	COCOMO < TDEV < פרויקטים, סוגים		PM = מאמץ הפיתוח הנדרש בחודשי אדם. נוסחאות – עמ' 33. טבלה של פילוג מאמץ – עמ' 34.
,137, 141, 142			הגדרות וסימונים – עמ' 141. דוגמאות – עמ' 142.
71	מתודולוגיות הכנת מפרט דרישות תוכנה < Data Flow < Oriented < Data Structure < Oriented < שיטות מפרט < פורמליות		במקום לנתח בעיה באמצעות מודל קלט-עיבוד-פלט (תזרים מידע) או באמצעות מודל הלקוח באופן בלעדי ממבני מידע מדרגים, (Object Oriented) OOA (Analysis) הכניסה מס' מושגים חדשים.
75,90			ראה תכן מונחה עצמים.
37	(תקנים (MIL) לפיתוח מע' תוכנה)		תיעוד תפעולי של המע' – מה הם כללי העבודה של הארגון.
,32, 34, 33	פרוייקטים, סוגים <		1. מערכת עיבוד במכלול (Batch). 2. מודלים מדעיים. 3. מודלים של מנהל עסקים. 4. מערכות מלאי וכדומה. 5. בד"כ עד 50,000 שורות קוד (50 KLOC).
94	(תכן מונחה עצמים) < הורשה/מדרג		צורת הורשה בין מחלקות, מופיע גם בשיטה הקלאסית לקביעת מחלקות ועצמים
			Part Of

104	Coad and Yourdon <	לפי Coad and Yourdon	
39	(תקנים (MIL לפיתוח מע' תוכנה)	קווי בסיס סופיים - אחרי שהמוצר מוגמר בסיום הפרויקט	PB Product Baseline
121	(בקרת שינויים) CCB <	בראשות מנהל הפרויקט. בסמכותה לאשר שינויי מסמכים (ושינויי התוכנה הנובעים מכך) שאינם משנים עלויות ולוחות זמנים.	PCCB
,39 ,42 ,112 113	(סקר)	בתום שלב Top level design (תכן העל). סקר זה מתבצע על ה-SDD ומובצע על כל מרכיב תוכנה עיקרי (CSCI או Sub level CSCI). בסקר זה, וסקר ה-CDR, המתבצעים בהשתתפות הנהלת המפתח ובד"כ גם בהשתתפות הלקוח, נסקרים מסמכי התכן באופן ביקורתי וניתן אישור להמשיך לתכן המפורט או לקידוד בהתאמה, אך תיתכן גם דרישה לשיפורים ולסקר חוזר. המטרה של סקר זה: הערכת ואישור של התקדמות, עקביות ונכונות גישת התכן הנבחרת למילוי דרישות הביצוע המפורטות במפרט הפונקציונלי. רשימת מה שנסקר - עמ' 112. ה-PDR חייב להיות מאושר ע"י הלקוח המאשר להמשיך ל-Detailed Design.	PDR סקר תכן מקדים <u>Preliminary Design Review</u>
120	(סוג של תחזוקת תוכנה) תוכנה, תחזוקה <	הוספת יכולת חדשה, הרחבת תפקודים, שיפור פונקציות קיימות.	Perfective Maintenance
35		Program Evaluation & Review Technique. מאפשר גם לראות מיידית מהו critical path-ומה ניתן לבצע במקביל. דומה ל-GANTT.	PERT
96	(מושגים מתקדמים תכן מונחה עצמים) מודל העצמים <	1. היכולת לקבל צורות שונות. למשל: רוטינת הדפסה יכולה להדפיס טקסט או גרפיקה. עצם של נהג המתקרב לצומת יכול להגיב לאור אדום/צהוב/ירוק. 2. מושג ב-Type Theory לפיו שם יכול לציין עצמים של מס' מחלקות שונות שיש להם superclass משותף. לפיכך, כל עצם בעל אותו שם מסוגל להגיב לקבוצה משותפת של פעולות בדרכים שונות. למשל: מחסנית שיכולה להיות ממומשת בדרכים שונות.	Polymorphism
120	(סוג של תחזוקת תוכנה) תוכנה, תחזוקה <	שינוי התוכנה כדי לשפר את התחזוקה שלה, ב-3 הסוגים הנ"ל.	Preventive Maintenance
100	(תכן מונחה עצמים) מחלקות ועצמים <	שיקולים בבחירת מחלקות ועצמים - הימנעות הן ממחלקות ועצמים מורכבים והן מפעולות מורכבות.	Primitiveness
97	מנשק המחלקה <	נגיש רק למחלקה עצמה ולמחלקות החברים.	Private
97	מנשק המחלקה <	נגיש רק למחלקה עצמה, ל-subclass שלה ומחלקות החברים.	Protected
121	(בקרת שינויים) ECP < בקרת תצורה <	Program Trouble Report - בקשות לשינויים מתקבלות כ-PTR - אם השינוי הנדרש בתוכנה בגלל תקלה/טעות. יכול להיות שכתוצאה מ-PTR נדרש ECP.	PTR
97	מנשק המחלקה <	נגיש לכל ה-clients שלו.	Public
30		מודל שמבוסס על שורות קוד.	Putman Estimation Model
126	(מדדי איכות תוכנה)	רשימה - עמ' 126.	Quality Factors
125	(מדדי איכות תוכנה)	רשימה ותרשים - עמ' 125.	Quality Metrics
,20 44	מערכת תוכנה, שלבי פיתוח, פיתוח מערכת, אסטרטגיות Grand Design Incremental Evolutionary <	מעבר לשפת תכנות חדשה, לחומרה חדשה או למתודולוגיות תכן חדשה (למשל OO), לאפשר הוספת יכולות או שינויים מהותיים למערכת קיימת. קובעים את הדרישות ומעדכנים על-פיהם את המע'.	Reengineering
97	מודל העצמים Sharing Copying/Cloning Adjusting <	3 צורות: 1. Sharing. 2. Copying/Cloning. 3. Adjusting.	Reuse
28,29	(ניהול סיכונים)	Risk Mitigation Monitoring and Management - צמצום, בקרה וניהול סיכונים. כל הסיכונים שמעל קו החיתוך שנקבע ע"י מנהל הפרויקט מוכנסים לתוכנית RMMM. דוגמה - עמ' 29.	RMMM
104	(תכן מונחה עצמים) מחלקות ועצמים, השיטה הקלאסית <	הגדיר שיטה לקביעת מחלקות ועצמים: • אנשים: בנ"א המבצעים תפקיד מסוים. כלומר, אם רוצים לסווג אנשים אזי נחפש תפקידים של אנשים ולפי זה נסווג אותם. • מקומות - אזורים המוקצים לאנשים או חפצים. • חפצים - ישויות גשמיות או מוחשיות. • ארגונים - אוסף מאורגן של אנשים, משאבים, מתקנים ויכולות בעלי מטרה מוגדרת. • תפיסות - עקרונות או רעיונות שאינם מוחשיים המשמשים לפעולות עסקיות ו/או לתקשורת. • מאורעות - דברים שקורים בד"כ במקום ובזמן מסוים או כצעדים באופן סדרתי.	Ross

105	(תכן מונחה עצמים) מחלקות ועצמים, השיטה ההתנהגותית	מציעים לזהות עצמים ומחלקות מתפקידי המע': תחילה מנסים להבין מה קורה במע' והיזמים והמשתתפים בעלי תפקידים ראשיים מזוהים כעצמים. תפיסה זו קרובה לרעיון של function points כאשר FP היא פונקציה עסקית אחת של משתמש סופי, כגון: קלט, פלט, שאילתה וכו'.	Rubin & Goldberg
121	(בקרת שינויים) CCB	הרמה הנמוכה ביותר. בראשות מנהל התוכנה. בסמכותה לאשר שינויי תוכנה, בעיקר לתיקון טעויות.	SCCB
37	(תקנים (MIL) לפיתוח מע' תוכנה)	אם המע' החדשה הולכת לעבוד על mainframe אזי מפעילי ה-mainframe צריכים לדעת שמע' חדשה מצטרפת, אילו הודעות היא שולחת וכו' – מדרך זה מתאר איך להפעיל את המע' החדשה.	SCOM Software Center Operator Manual
,37 ,42,43 ,77 ,112 113	(תקנים (MIL) לפיתוח מע' תוכנה)	תיאור תכן המע'. שייך לתיק העיצוב בנוהל מפת"ח. SDD1 – preliminary design, SDD2 – detailed design	SDD Software Design Description
37	(תקנים (MIL) לפיתוח מע' תוכנה)	תוכנית פיתוח של התוכנה – Software Development Plan.	SDP
,39 55, 42	(סקר)	בתום שלב הנדסת המע' אם עובר בשלום אז נקבע Functional Baseline . עורכים סקר זה בתום ניתוח המע' כדי לוודא: 1. האם המורכבות הפונקציונלית של המע' תואמת הערכות העלויות, ה"ל"ז וסיכוני הפיתוח. 2. האם המנשקים בין מרכיבי המע' ומנשקים חיצוניים מוגדרים בפירוט מספק (האם ארכיטקטורת המע' מוגדרת היטב). 3. האם נושאי ביצועים, אמינות ותחזוקה נלקחו בחשבון באופן מספק. 4. האם ה-SSS (System Specification) נותן מספיק בסיס לשלבי הנדסת החומרה והנדסת התוכנה שיבואו. 5. האם נשקלו חלופות למימוש.	SDR System Design Review
32	(סוגי פרוייקטים ב- COCOMO1) פרוייקטים, סיווג	1. מערכות עיבוד תנועות רגילות, למשל: בכספומט: הכנסת כרטיס, הוצאת כרטיס. 2. מערכות פיקוח ייצור חכמות. 3. מערכות הפעלה חדישות. 4. בד"כ עד 300,000 שורות קוד (300 KLOC).	Semidetached
98	(סוג סנכרון) Synchronization	בכל עת קיים עצם פעיל אחד בלבד.	Sequential
97	(תכן מונחה עצמים) Link	עצם שאיננו מפעיל עצמים אחרים והוא רק מופעל ע"י עצמים אחרים.	Server
97	(צורה של reuse מושגים מתקדמים במונחה עצמים) Reuse	למשל: שימוש בטלפון ע"י אנשים שונים או רוטינה הבודקת חוקיות של מס' זיהוי.	Sharing
104	(תכן מונחה עצמים) מחלקות ועצמים, השיטה הקלאסית	הגדירו שיטה לקביעת מחלקות ועצמים: • דברים מוחשיים, כגון: מכונית, ציונים, נתוני חום. • תפקידים, כגון: מורה, אמא, פוליטיקאי. • מאורעות, כגון: נחיתה, פסיקה, בקשה. • פעולות גומלין, כגון: הלוואה, פגישה, חצייה.	Shlaer & Mellor
111	(מנשק משתמש)	שימוש בצבעים במנשק משתמש - ראה צבעים	Shneiderman B.
37	(תקנים (MIL) לפיתוח מע' תוכנה)	מתאר ק/פ של המע' – נחוץ כאשר הק/פ מורכב/מסובך.	SIOM Software Input / Output Manual
37	(תקנים (MIL) לפיתוח מע' תוכנה)	איך מתקנים את המע'.	SIP Software Installation Plan
,37 38,39 115	(תקנים (MIL) לפיתוח מע' תוכנה)	מפרט מוצר תוכנה - Software Product Specification. נכתב לאחר הקידוד. הכי חשוב בו הוא ה-listings (תדפיסים) של ה-source code. מכיל גם הפניות איך הקוד נגזר מהעיצוב.	SPS Software Product Specification
,121 ,122 ,123 124	Verification Validation בקרת תצורה איכות תוכנה, מדדים	הבטחת איכות מחייבת בקרה על תהליך ביצוע השינויים כדי לוודא ששינויים שאושרו אכן בוצעו והביצוע היה עפ"י כללי הנדסת התוכנה וכל השינויים הנגררים גם כן בוצעו (למשל: האם כתוצאה משינוי בתוכנה שינוי גם את התיעוד ושינוי גם את מסמכי התכן). 2 מושגים בסיסיים להבטחת איכות: 1. Verification. 2. Validation. הפעילויות העיקריות של SQA הן: 1. יישום שיטות טכניות. 2. ניהול סקרים פורמליים (אנשי אבטחת איכות מקפידים שהסקרים מתנהלים לפי הכללים). 3. בדיקות תוכנה. 4. כפיית תקנים. 5. בקרת שינויים CM.	SQA הבטחת איכות תוכנה

		<p>6. מדידות (ביצועים). 7. רישומים ודיווחים.</p> <p>צוות ה-SQA מייצג את הלקוח ולפיכך נקודת המבט שלו היא נקודת המבט של הלקוח. הצוות מזהה מדדי איכות נדרשים ובודק אם התוכנה עונה על דרישות האיכות. כמו כן עוקב הצוות אחרי כל שלבי הפיתוח ומוודא שהפיתוח מתבצע עפ"י התקנים שנקבעו. לבסוף, הצוות מבקר את כל מבחני הקבלה. הצוות מלווה את הפרוייקט מתחילתו ועד תום החוזה. הבטחת איכות תוכנה היא מיפוי ההוראות הניהוליות ומשמעת הבטחתה האיכות, על המרחב הניהולי והטכנולוגי של הנדסת התוכנה. אם מיפוי זה מושג בהצלחה אזי התוצאה המתקבלת היא הנדסת תוכנה בשלה. תרשים – עמ' 124.</p>	
39 55	(סקר)	<p>בתום שלב הצגת הדרישות של הלקוח. רצוי לערוך סקר זה בתום ניתוח הדרישות כדי לוודא:</p> <ol style="list-style-type: none"> האם הוגדר צורך מבוסס, האם יש צידוק למע'. האם הגוף המוגדר (או השוק) צריכים את המע' שנדרשה. האם נשקלו חלופות למע' ואלו. האם קיימים משאבים לביצוע המע'. האם אילוץ העלות והל"ז הם ריאליים. האם קיימים סיכונים פיתוח. 	<p>SRR <u>System Requirements Review</u></p>
37,38 42 72,43 79		<p>מפרט דרישות התוכנה. יחד עם IRS הם מהווים המקבילה לתיק האפיון בנוהל מפת"ח (Software Requirement Analysis). מקובל שלכל CSCI יש SRS שלו.</p>	<p>SRS <u>Software Requirements Specification</u></p>
37	(תקנים (MIL) לפיתוח מע' תוכנה)	<p>System/Subsystem Design Description. נותן טרמינולוגיה שבה משתמשים במע' (למשל: מינוחים מקצועיים בחדר טיפול נמרץ).</p>	<p>SSDD</p>
39 73,72	(סקר)	<p>בתום שלב ניתוח דרישות התוכנה / אפיון – תיק האפיון. אם מאשרים אותו מקבלים Allocated Baseline. הסקר מתבצע על IRS, SRS ו-STP. המטרה: הצגת והערכת ארכיטקטורות התוכנה בכללותה, כולל הקצאת הדרישות הפונקציונליות. רשימה של מה נסקר – עמ' 73-72. עם אישור ה-IRS, SRS וה-STP (= תיק עיצוב, תיק מבדקים) נקבע ה-Allocated Baseline לפרוייקט.</p>	<p>SSR <u>Software Specification Review</u></p>
42,37 55,72	(תקנים (MIL) לפיתוח מע' תוכנה)	<p>מהווה יחד עם ה-STP את ה-System Design. מפרט של מע'/תת-מע' שמתאר את המע' בתום שלב ניתוח המע'. מכיל את ה-CSCI ו-HWCI ומנסקים בין ה-CSCI's ובין ה-HWCI's. אחרי SSS ניתן לפתח 2 מסמכים שהם בין שלב הנדסת המע' לבין שלב פיתוח התוכנה: STP, SDP. בתום ה-SDR הלקוח מאשר את ה-SSS (לעתים אחרי הכנסת שינויים עפ"י המתעורר מהסקר) והוא משמש כ-Functional Baseline.</p>	<p>SSS <u>System/Subsystem Specification</u></p>
95	(עקרונות מודל העצמים)	<p>ה-type של כל המשתנים והביטויים נקבעים בזמן הקומפליציה.</p>	<p>Static Binding – Early Binding</p>
63	מודל התנהגותי	<p>State Transition Diagram – מהווה הרחבה של אס"ד.</p>	<p>STD</p>
37 42 43,77 117		<p>Integration & Test. שייך לתיק סיכום מבדקים בנוהל מפת"ח. תיאור מבדקי תוכנה – איך יערכו מבדקי המערכת. מפרט לפרטי פרטים איך נערך כל מבדק (איפה, איזה ציוד, מי המשתתפים, מי מכניס נתונים, אילו תוצאות מצפים לקבל, מה עושים אם מקבלים תוצאות אחרות וכו').</p>	<p>STD <u>Software Test Description</u></p>
37 42 43 72,55 117	Drivers Stubs	<p>מהווה יחד עם ה-SSS את ה-System Design. שייך לתיק סיכום מבדקים בנוהל מפת"ח. מבצע אותו מנתח המע'. תוכנית מבדקי התוכנה. ב-SSS יודעים מה המע' צריכה לבצע מבחינה פונקציונלית ומבחינת התוכנה ונקבעת כבר מה צריכה להיות פונקציונליות המע' ולכן ניתן לכתוב כבר את ה-STP. בתום גיבוש ה-SSS ואפילו לפני הצגתו ואישור פורמלי מכינים את ה-STP אשר גם הוא חייב להיות מאושר ע"י הלקוח. ייתכן שבזמן ניתוח דרישות התוכנה וכתבת ה-SRS מעדכנים/משלימים את ה-STP. ב-STP קובעים איך ייעשו המבדקים ובודקים שאכן הפונקציונליות ממומשת כמו שצריך. עמ"נ שה-STP יהיו יעילים חייבים לתכנן אותם היטב ולהתחיל בתכנון בשלב מוקדם של הפרוייקט. רצוי להתחיל בתכנון STP מיד לאחר ההשלמה (הראשונה) של ה-SSS ולעדכן אותו במידת הצורך עם השלמת ה-SRS. חייבים לכלול בו:</p> <ol style="list-style-type: none"> שלבי הבדיקות העיקריים חייבים להיות מזהים היטב ומסודרים. היחסים בין קריטריוני האימות חייבים לבטא את דרישות התוכנה: לא להוסיף נתאים שלא מופיעים בדרישות ולבודק את כל התנאים שכן הופיעו. פונקציות מרכזיות של המע' חייבות להיות מודגמות בזמן מוקדם של הפרוייקט. תוכנית הבדיקות חייבת להיות תואמת לתוכנית הפרוייקט. לוח הזמנים לבדיקות חייב להיות מוגדר בביור. המשאבים והכלים הנחוצים לבדיקות חייבים להיות מזהים וניתנים להשגה. 	<p>STP <u>Software Test Plan</u></p>

		<p>7. חייבים להגדיר מנגנון רישום ודיווח של הבדיקות.</p> <p>8. תוכניות מבחן (drivers) וגדמים (stubs) חייבים להיות מזוהים ופיתוחם צריך להיות מתוזמן.</p> <p>9. הבדיקות חייבות לכלול מבחני מעמס לתוכנה: עמידה בתנאים מקס' מסוימים המופיעים בדרישות התוכנה.</p>	
37,43 117	<p>(תקנים (MIL) לפיתוח מע' תוכנה)</p>	<p>דו"ח מבדקי תוכנה – מסכם את המבדקים שנערכו בקבלת המע'.</p> <p>שייך לתיק סיכום מבדקים בנוהל מפת"ח.</p>	<p>STR Software Test Report</p>
37	<p>(תקנים (MIL) לפיתוח מע' תוכנה)</p>	<p>תוכנית מעבר של התוכנה: יוצאים מתוך הנחה שבארגון קיימת כבר תוכנה שעושה חלק מהפונקציונליות של המע' וכן מע' אחרות שיש להן אינטראקציה עם המע' הישנה – בשלב זה עושים את המעבר למע' החדשה.</p>	<p>STrP Software Transition Plan</p>
72		<p>ה-stub הוא תוכנית שקולטת את הפלטים של CSCI מסוים כדי לבדוק אם הוא ביצע את כל הפעולות שלו.</p>	<p>Stubs גדמים</p>
100	<p>(תכן מונחה עצמים)
< מחלקות ועצמים</p>	<p>המידה בה מחלקה או עצם כולל מספיק מאפיינים של ההפשטה כדי לאפשר הידודיות משמעותית ויעילה.</p>	<p>Sufficiency</p>
37	<p>(תקנים (MIL) לפיתוח מע' תוכנה)</p>	<p>מדריך למשתמש – Software User Manual.</p>	<p>SUM</p>
37	<p>(תקנים (MIL) לפיתוח מע' תוכנה)</p>	<p>Software Version Description.</p> <p>מסמך שמרכז את הגרסאות השונות של התוכנה ושל ה-tests. נכתב במקביל ל-SSS.</p>	<p>SVD</p>
98	<p>(סנכרון)
< Link
< Message
< Sequential
< Guarded
< Synchronous
< Balking
< Timeout
< Asynchronous</p>	<p>כאשר עצם אחד מעביר message לעצם אחר דרך link אזי 2 עצמים אלו הם synchronized. ה-synchronization הוא מאחד הסוגים העיקריים:</p> <p>1. Sequential.</p> <p>2. Guarded.</p> <p>3. Synchronous.</p> <p>סוגי synchronization:</p> <p>1. Synchronous.</p> <p>2. Balking.</p> <p>3. Timeout.</p> <p>4. Asynchronous.</p>	<p>Synchronization</p>
98	<p>(סוג סנכרון)
< Synchronization</p>	<p>קיימים מס' תהליכים פעילים וה-suppliers מבטיחים מניעה הדדית.</p>	<p>Synchronous</p>
98	<p>Synchronization
< Messages</p>	<p>פעולה מתחילה רק כאשר השולח יזם אותה והמקבל מוכן לקבל את ה-message. שניהם ימתנו ללא הגבלת זמן עד ששני הצדדים יהיו מוכנים להמשיך (בדומה לשיחת טלפון: א' מחייג לב', ב' עונה והם מדברים).</p>	<p>Synchronous</p>
34	<p>COCOMO
< MM
< פרויקטים, סוגים</p>	<p>זמן פיתוח. התוצאה בחודשים. נוסחאות + דוגמה – עמ' 34.</p> <p>טבלה של פילוג זמנים – עמ' 34.</p>	<p>TDEV</p>
98	<p>(סוג סנכרון)
< Synchronization
< Synchronous</p>	<p>כמו synchronous רק שהשולח ימתין זמן מוגדר למקבל להיות מוכן (למשל: א' מחייג לב', ממתין 2 צלצולים ואם אין תגובה מנתק).</p>	<p>Timeout</p>
79 81,80	<p>(תכן מונחה זרימת נתונים)
< Data Flow
< Oriented Design</p>	<p>= תנועה. משתמשים כאשר זרימת המידע מתפצלת למס' רב של מסלולים. תרשים – עמ' 80.</p>	<p>Transaction Analysis</p>
79 80 86,81	<p>(תכן מונחה זרימת נתונים)
< Data Flow
< Oriented Design</p>	<p>משתמשים בו בזרימת מידע כאשר קיימת אבחנה ברורה של הגבולות בין נתונים נכנסים ונתונים יוצאים. תרשים – עמ' 79.</p>	<p>Transform Analysis</p>
39	<p>(סקר)</p>	<p>Test Readiness Review. לפני ה-CSCI Testing (= לפני בדיקות המע'). סקר למוכנות המבדקים.</p>	<p>TRR</p>
108 158		<p>תקן סימון עבור תבנית מע' עצמים מונחים. ה-UML הוא רק תקן סימון ובעיקרו הוא מגדיר מס' תרשימים אשר ניתן לשרטט ולתאר את המע' ואת משמעותם של התרשימים. ה-UML לא מתאר את התהליך בו משתמשים עמ"נ לבנות את התוכנה. הוא מעין תיאור תהליך או שיטה אשר תכלול רשימת משימות אשר צריכות להיעשות, התוצרים שיתקבלו, סוג המיומנות הנדרש עבור כל משימה וכו'.</p> <p>דוגמאות – עמ' 109-110.</p>	<p>UML</p>
105 106	<p>(שיטה לקביעת מחלקות ועצמים בתכן מונחה עצמים)</p>	<p>ראה מחלקות ועצמים, Use-Case.</p>	<p>Use-Case</p>
100	<p>(תכן מונחה עצמים)
< מחלקות, יחסים בין</p>	<p>מקביל לקשר client/supplier בעצמים. זהו עידון של יחס מסוג association. ביחס זה ברור מי המפעיל ומי המופעל.</p>	<p>Using</p>
123	<p>SQA <</p>	<p>Are we building the right project?</p>	<p>Validation</p>
123	<p>SQA <</p>	<p>Are we building the project right?</p>	<p>Verification</p>
116	<p>(בדיקות תוכנה)
< בדיקות פנימיות</p>	<p>נערך כדי לוודא שהמימוש עונה על הדרישות. נעשה ע"י ראש צוות או ע"י קבוצות שלמות. הלקוח לא מעורב בזה.</p>	<p>Walk-Through</p>

		המטרה: לגלות בעיות ולענות על "what if". מה התוכנה עושה ולא איך. רשימה של מה שבודקים – עמ' 116.	
69,70 83	(תכן מונחה מבנה נתונים) Data Structure Oriented	שיטה לכתובת מידע העובר בין יצרנים/צרכנים בתרשימי DSSD או במתודולוגית Warnier-Orr דוגמה בעמ' 70.	Warnier-Orr תרשים ישויות
99	(תכן מונחה עצמים) מחלקות, יחסים בין	דהיינו, יחס "Part Of", למשל: המחלקה "עלי כותרת" היא חלק של המחלקה "פרחים".	Whole/Part
105	(תכן מונחה עצמים) מחלקות ועצמים, השיטה ההתנהגותית	לפי Wirfs-Brock נקבעים העצמים עפ"י אחריות. האחריות של עצמים הם כל השירות שהעצם מספק. עצמים בעלי אחריות משותפת מאוגדים למחלקה ומחלקות כאלו ל-superclass.	Wirfs-Brock
17, 15	(שיטות פיתוח מערכת) מערכת תוכנה, שלבי פיתוח מנשק אנוש מחשב 4GT	נבנה בשלב הנדסת המערכת כאשר הדרישות לא מוגדרות היטב, כשקיימת אי-ודאות לגבי יעילות אלגוריתם, כשרוצים לבדוק התאמת מערכת הפעלה או מסד נתונים או כשרוצים להגדיר את מנשק אנוש מחשב (HCI). אבטיפוס הוא תהליך המאפשר למפתח התוכנה ליצור דגם של התוכנה או של רכיב מסוים שלה שצריך להיבנות. 3 צורות לאבטיפוס: 1. אבטיפוס מנייר או מודל המבוסס על PC המתאר פעולת גומלין של אנוש-מחשב בצורה שממחישה למשתמש כיצד פועלים יחסי הגומלין. 2. אבטיפוס פעיל ומעשי שמיישם חלק מהפעולות הנדרשות מהתוכנה המבוקשת. 3. תוכנית קיימת המבצעת חלק מהפעולות או את כל הפעולות המבוקשות, אך בעלות מאפיינים אחרים שאינם מהווים תחליף לפיתוח. גישת האבטיפוס מתחילה באיסוף הדרישות. מפתח התוכנה והלקוח מגדירים את היעדים הכלליים, מזהים את הדרישות הידועות ומדגישים תחומים שבהם צריך הגדרה נרחבת יותר. מתהווה תרשים מהיר שמתמקד בהצגת ההיבטים של התוכנה לעין המשתמש. תרשים זה מוביל לבניית האבטיפוס. האבטיפוס מוערך ע"י המשתמש/לקוח ומטרתו להציב דרישות לתוכנה עמ"נ שיפותחו. תרשים – עמ' 15.	אבטיפוס Prototyping
76	(תכן מבנה) תכן ארכיטקטורה ליכוד צימוד	מודול מבצע רק פונקציונליות מסוימת. תכונה זו ממדת ע"י 2 קריטריונים: ליכוד וצימוד.	אי תלות פונקציונלית
8-11	(משבר התוכנה)	חשיבות האיכות, מחיר איכות לא מספקת, כולל דוגמאות	איכות
,124 ,125 ,126 127	(מדדי איכות תוכנה) SQA Quality Metrics Quality Factors	תוכנה נקראת איכותית אם היא מתאימה ל: 1. דרישות פונקציונליות ודרישות ביצוע מפורשות. 2. תקני פיתוח מתועדים במפורש. 3. מאפיינים מובהקים שמצפים מכל תוכנה שפותחה מקצועית.	איכות תוכנה, מדדים
,127 ,128 ,129 130	(מדידת איכות תוכנה) SQA Helstead's Complexity Measure McCabe's Complexity Measure DSQI אינדקס בשלות תוכנה	קביעת המדדים היא סובייקטיבית. קביעת ערכים מספריים למדדים דורשת מומחיות, כך גם הערכת איכות תוכנה טובה מבוססת על סובייקטיביות ומומחיות. 2 מטריקות שמודדות את המורכבות של תוכנית: 1. Helstead's Complexity Measure – ראה עמ' 128. 2. McCabe's Complexity Measure – ראה עמ' 128. 2 מדדים נוספים: 1. DSQI – ראה עמ' 129. 2. אינדקס בשלות תוכנה – ראה עמ' 130.	איכות תוכנה, מטריקות
130	(מדידת איכות תוכנה) איכות תוכנה, מטריקות	IEEE. נותן אינדיקציה ליציבות של מוצר תוכנה. נוסחה – עמ' 130.	אינדקס בשלות תוכנה
103	(תכן מונחה עצמים) סיווג מחלקות ועצמים	סיווג עפ"י תפיסה. גישה זו נובעת מהניסיונות לייצוג ידע. המחלקות נוצרות ע"י ניסוח תיאור תפיסתי עבורן וסיווג הישויות עפ"י תיאורים אלו. למשל: ספרים יכולים להיות מסווגים עפ"י ספרי קודש, ספרי מתח, בישול וכדו'. אין מדד אובייקטיבי לקבוע למשל את מידת "המתח" של ספר או מידת היות שיר "שיר אהבה". ההשתייכות למחלקה היא עפ"י best fit ולא absolute (absolute – מבחון).	אישיכול תפיסתי
,130 ,131 ,132 ,133 ,134 ,135 136	SQA המודל הבסיסי המודל הלוגריתמי-פואסוני	נוסחאות – עמ' 130. 2 מודלים: 1. המודל הבסיסי – עמ' 131-132. 2. המודל הלוגריתמי-פואסוני – עמ' 132-133. גרף השוואה בין המודלים – עמ' 134 + ייעוד של כל מודל. מדד להפעלה מבצעית - עמ' 135. מדדים נוספים שהועתקו ממדי אמינות לחומרה: MTBF – Mean Time Between Failures. MTTR – Mean Time To Repair. MTBF = MTTF + MTTR נוסחה – עמ' 135. העתקה זו נתונה לביקורת כי בעוד בחומרה רכיב פגום מוחלף ברכיב תקין, בתוכנה תיקון תקלה עלול לחייב שינוי בתכן שיכול לגרום לתקלות חדשות.	אמינות

53	<ul style="list-style-type: none"> חלקי מע' המחשב מנשק אנוש-מחשב 	<p>העיקרון הקובע הוא ידידות למשתמש.</p>	אנשים
78	<ul style="list-style-type: none"> ארכיטקטורת תכן מודל המאגר מערכות Event Driven 	<p>מודל מע' מבוזר המראה כיצד העיבוד והנתונים מבוזרים על מס' מרכיבים. המרכיבים העיקריים הם:</p> <ol style="list-style-type: none"> 1. קבוצה של שרתים שכל אחד בפני עצמו מספק שירות ספציפי, כגון: הדפסה, ניהול קבצים/נתונים, קומפילציה וכו'. 2. קבוצה של לקוחות המבקשים שירותים אלו. 3. רשת תקשורת המאפשרת גישת הלקוחות לשרתים. <p>יתרונות:</p> <ol style="list-style-type: none"> 1. העברת וביזור נתונים היא ישירה. 2. ניצול יעיל של רשת התקשורת. 3. עלות החומרה עשויה להיות זולה יותר (קונים הרבה רכיבים, כי כל רכיב מבצע פונקציה מאוד מסוימת, אולם הרכיבים עצמם זולים). 4. קל להוסיף שרתים חדשים או לשדרג שרתים קיימים. <p>חסרונות:</p> <ol style="list-style-type: none"> 1. אין מודל נתונים משותף ולכן כל תת-מע' תשתמש בארגון נתונים שונה והחלפת נתונים עלולה להיות בלתי יעילה (כל שרת ושרת מטפל בנתונים באופן שונה ויכול להיות ששרת אחד, למשל, ממיינ נתונים לפי ת.ז. ואילו השני לפי מס', אז יש קושי בהחלפת הנתונים ביניהם). 2. יתירות של ניהול בכל שרת. 3. אין רישום מרכזי של שמות ושירותים, דבר המקרשה לגלולת אלו שרתים ושירותים נגשים. 	<p>ארכיטקטורת שרת-לקוח</p>
77, 79, 78	<ul style="list-style-type: none"> מודל המאגר ארכיטקטורת שרת לקוח מערכת Event Driven 	<ol style="list-style-type: none"> 1. מודל המאגר. 2. ארכיטקטורת שרת/לקוח. 3. מערכת Event Driven. 	ארכיטקטורת תכן
14, 116, 117, 118			בדיקה
117, 118	<ul style="list-style-type: none"> תוכנה, בדיקות STP STD STR 	<p>בדיקות אלו מתבצעות אחרי הבדיקות הפנימיות. הבדיקות הנ"ל מתבצעות עפ"י ה-STP וה-STD. הבדיקות מתבצעות בד"כ בשלבים כאשר בתחילה בודקים תפקודים מסוימים בודדים, אח"כ תת-מע' ברמות שונות ועד בדיקת המע' כולה. כל בדיקה חיצונית מתוארת ב-STR. לפני הבדיקה עושים Test Review: סקר הבדיקות מתבצע ב-2 חלקים:</p> <ol style="list-style-type: none"> 1. הערכת מוכנות התוכנה לביצוע הבדיקות. 2. הערכת "מספיקות" הבדיקות והתוצאות המדווחות. 	בדיקות חיצוניות
116, 118	<ul style="list-style-type: none"> תוכנה, בדיקות Walk-Through Inspections סקרי סיכום 	<ol style="list-style-type: none"> 1. Walk-Through. 2. Inspections. 3. סקרי סיכום. <p>דוגמה: עמ' 117-118.</p>	בדיקות פנימיות (סקרים הנדסיים)
120, 121, 122	<ul style="list-style-type: none"> הנדסת תוכנה CI PTR ECP SQA 	<p>תהליך הנדסת התוכנה מייצר:</p> <ul style="list-style-type: none"> • תוכניות מחשב הן כ-Source והן כ-Executable. • מסמכי תיעוד התוכניות, המדריכים והבדיקות. • מסמכי הניתוח והתכן של המערכת. <p>כל מע' משתנה במשך מהלך חייה והשינויים קורים (עוד בזמן הניתוח) ועד סוף מחזור חייה. בקרת תצורה היא סדרת פעילויות שנועדה לנהל את השינויים תוך הבטחת איכות התוכנה. בקשות לשינויים מתקבלות כ-PRT או כ-ECP.</p>	בקרת תצורה – CM
54	<ul style="list-style-type: none"> חלקי מע' המחשב מנשק אנוש-מחשב 	<p>פעמים רבות הוא הגורם החשוב ביותר להצלחת/כשלון מערכת. בזמן ניתוח הדרישות חיוני להתחשב בגורם האנושי ולעתים להיעזר במומחים במדעי החברה לביצוע ניתוח Ethnography שיעקרו התבוננות בסביבת העבודה במהלך הפעילות היומיומית השוטפת ולראות איך המחשב משתלב בחיי הארגון.</p>	גורם אנושי
14	<ul style="list-style-type: none"> מערכת תוכנה, שלבי פיתוח 	<p>בשלב זה נקבע ה"מה". הדרת המע' = הנדסת המע' + ניתוח = אפיון.</p>	הגדרה
90	(תכן מונחה עצמים)	מושגים בתכן מונחה עצמים	הדגם – Instance
90	(תכן מונחה עצמים)	מושגים בתכן מונחה עצמים	הורשה – Inheritance
94	<ul style="list-style-type: none"> (תכן מונחה עצמים) מודל העצמים 	<p>מדרג הוא דירוג או סידור של הפשטות.</p> <ol style="list-style-type: none"> 2. ההיררכיות החשובות ביותר במע' מורכבת הן: <ol style="list-style-type: none"> 1. מבנה המחלקה – היררכית "Is-A". 2. מבנה העצם – היררכית "Part-Of". <p>הורשה היא יחס בין מחלקות שבו מחלקה אחת יורשת את המבנה או ההתנהגות המוגדרים במחלקה אחרת או במספר מחלקות אחרות. הורשה מגדירה היררכית Is-A בין מחלקות בה subclass יורש מאחד או יותר superclasses כלליים יותר. Subclass מייחד את ה-superclass שלו ע"י הרחבה או הגדרה מחדש של המבנה</p>	הורשה / מדרג

		וההתנהגות הקיימים. למשל: Person is a faculty/admin/student (person של הייחוד של).	
98	(יחסים בין עצמים) Link < Message <	בין 2 עצמים הקשורים ע"י link קיימת היראות כדי להעביר messages מאחד לשני. היראות זו היא מאחד מ-4 הסוגים הבאים: 1. The supplier object is global to the client. 2. The supplier object is a parameter to some operation of the client. 3. The supplier object is part of the client object. 4. The supplier object is a locally object in some operation of the client.	היראות
,131 ,132 134	(מדדים לאיכות תוכנה) אמינות <	מומלץ להשתמש במודל זה תמיד (למשל: במע' הנה"ח), למעט כאשר ההפעלה המבצעית של המע' היא מאוד לא אחידה. נוסחאות ודוגמאות – עמ' 134, 132, 131.	מודל הבסיסי
,132 ,133 134	(מדדים לאיכות תוכנה) אמינות <	מומלץ להשתמש במודל זה כאשר ההפעלה המבצעית של המע' היא מאוד לא אחידה. למשל: מדדי חולה בטיפול נמרץ (כל פעם יכול להיות פרמטר אחר שיוצא מהגבולות המותרים) או מע' שמקבלת קלט מגלאים שמגלים תופעות שקורות מחוץ למע' ואין שליטה על תופעות אלו, כמו למשל בקרת טילים). נוסחאות ודוגמאות – עמ' 134-132.	מודל הלוגריתמי פואסוני
17	המודל הספירלי <	Engineering - פיתוח המוצר ברמה הבאה (Next Level).	הנדסה
42, 7	מהנדס מע' מחשב <	פעילות של מידול ופיתוח בעיות במחשוב מערכות. תפקודי מע' נדרשים נחשפים, מנותחים ומוקצים לגורמי מע' אינדיבידואלים.	הנדסת מערכות ממוחשבות
14	מודל מפל המים <	ניתוח הדרישות והקצאת נתחי התוכנה. מה נבצע בחומרה (חומרה ייחודית לפרוייקט)? בתוכנה? האם יש צורך בקושחה? אילו מפעולות המערכת יבוצעו ע"י אנשים? מסמך ייזום	הנדסת מערכת
,8, 7 12	IEEE < ייצור < פרוייקטים < איכות מע' תוכנה < משבר התוכנה <	1. כינון ושימוש בעקרונות הנדסיים מבוססים כדי לקבל באופן כלכלי תוכנה אמינה ויעילה. 2. IEEE: יישום גישה שיטתית מיוסדת על כללים עבור פיתוח, תפעול ותחזוקה של תוכנה. 3. מחקר של גישות לפי (2). יתרונות הנדסת תוכנה: 1. ייעול ייצור (מערכות). 2. ייעול איכות. 3. צמצום עלות התחזוקה. הנדסת תוכנה תמיד כפופה לתכתיבי תקציב ולאילוץי ל"ז. אלה נקבעים ע"י הארגון שמפתח את התוכנה.	הנדסת תוכנה
12			הנדסת תוכנה, יסודות
76	(עקרונות תכן) תכן נתונים <	עיקרון זה מאפשר חלוקת מע' למודולים. עפ"י העיקרון, כל מודול מסתיר את המידע שבו ומסר למודולים האחרים רק את הנתונים הדרושים להם.	הסתרת מידע
94, 93	(עקרונות מודל עצמים) מודל העצמים <	תהליך חלוקת הרכיבים של הפשטה לאלו המהווים את המבנה ואת ההתנהגות שלה. ההסתרה מסייעת להפריד בין המנשק החוזי של הפשטה לבין היישום שלה. המימוש = "איך" – מוסתר. "מה" = החלק הגלוי.	הסתרת מידע
17	המודל הספירלי <	הערכת תוצאות הפיתוח ההנדסי ע"י הלקוח.	הערכת הלקוח
135	(מדדים לאיכות תוכנה)	אחרי מסירת התוכנה ללקוח נוסחאות לבדיקת אמינות.	הפעלה מבצעית
76	(עקרונות תכן) תכן נתונים <	התרכזות בעביה ברמה של הכללה מבלי להתייחס לפרטים בלתי רלוונטיים ברמה נמוכה יותר. בכל שלב של הנדסת התוכנה קיימת רמה שונה של הפשטה. בשלב הניתוח רמת ההפשטה היא הגבוהה ביותר והיא מתעדנת כשעוברים לשלבי תכן על ותכן מפורט עד שמגיעים בקוד לרמת ההפשטה הנמוכה ביותר.	הפשטה
75 93	(עקרונות מודל עצמים) מודל העצמים <	המאפיינים העיקריים של עצם המבדילים בינו לבין כל הסוגים האחרים של עצמים וע"כ מגדירים בצורה קולעת את הגבולות המושגיים ביחס לנקודת המבט של המתבונן. זהו התהליך של התמקדות במאפיינים העיקריים של עצם.	הפשטה
43		בקשה למכרזים. זהו חלק מנהלי ולא חלק של פיתוח תוכנה.	הצעות ספקים ומפרט RFP
7	מהנדס מע' מחשב <	לאחר שהתפקוד, האילוץ והמנשקים נתחמים עוברים למטלה שנקראת הקצאה. במצב זה כל תפקוד מוקצה לאחד או יותר מיסודות המערכת הכוללת (תוכנה, חומרה, אנשים וכו'). לעתים, הקצאות חלופיות מוצעות ומוערכות.	הקצאה
54	(חלקי מע' מחשב) התכנות, בדיקות <	בעיות חוקיות שיתעוררו מפיתוח המע' (בישראל: כולל בעיות יחסי עבודה).	התכנות חוקית
54	(חלקי מע' מחשב) התכנות, בדיקות < התכנות כלכלית < התכנות חוקית < חלופות <	1. האם הטכנולוגיה הדרושה קיימת? 2. מהם סיכוני הפיתוח? 3. האם המשאבים קיימים וזמינים? 4. בדיקה ע"י מודל מתמטי או פיזי של חלקי המע' העיקריים ואלו בעלי הסיכון הגבוה.	התכנות טכנית
54, 53	(חלקי מע' מחשב) התכנות, בדיקות < התכנות חוקית < התכנות טכנית < חלופות <	ניתוח עלות/תועלת: 1. יתרונות בחישובים מהירים ובהדפסות. 2. יתרונות באגירת נתונים. 3. יתרונות בחיפוש נתונים. 4. יתרונות מיכולת ביצוע סימולציות וחישובים מורכבים. 5. יתרונות מבקרת תהליכים ומשאבים.	התכנות כלכלית

		<p>6. עלויות הרכישה (חומרה). 7. עלויות התנעת המע' (startup). 8. עלויות פיתוח הפרוייקט. 9. עלויות תחזוקה שוטפת.</p>	
53	<p>(חלקי מע' מחשב) < התכנות כלכלית < התכנות טכנית < התכנות חוקית < חלופות</p>	<p>הנחה: זרישות הלקוח הוגדרו והובנו. סוגי בדיקות: 1. התכנות כלכלית. 2. התכנות טכנית. 3. התכנות חוקית. 4. חלופות.</p>	התכנות, בדיקות
95	(עקרונות מודל העצמים)	<p>התכונה של עצם באמצעות קיומו הוא מעבר לזמן ו/או למקום. דהיינו, הוא ממשיך להתקיים גם לאחר שהיוצר שלו הפסיק להתקיים ו/או מיקום העצם עובר ממרחב הכתובות בו הוא נוצר. למשל: עצם database oriented מתקיים גם כאשר התוכניות שהפעילו אותו הפסיקו להתקיים או במערכת מבוצרת ייתכן ועצם עובר ממעבד אחר למעבד אחר.</p>	התמדה
51, 50	<p>(חלקי מע' מחשב) < מערכת המחשב, < חלקי</p>	<p>מחשב, יחידות עיבוד (מהירות וגודל זיכרון), יחידות אחסון (דיסקים: קיבולת, מהירות גישה), ציוד היקפי, יחידות ק/פ. יש חשיבות ל: 1. עלות (רכישה ותחזוקה) – התחזוקה עלולה להיות יקרה מעלות הרכישה. 2. שיקולים לוגיסטיים (אמינות, מקום, סוג קירור, קלות תחזוקה). 3. קווי תקשורת. 4. סוג המשתמשים.</p>	חומרה
54	<p>(חלקי מע' מחשב) < התכנות, בדיקות < התכנות כלכלית < התכנות טכנית < התכנות חוקית</p>	<p>1. חלופות חומרה. 2. חלופות מימוש חומרה/תוכנה, כולל חלופות שימוש בקושחה. 3. חלופות מע' הפעלה, מסד נתונים, מהדרים, תוכנות מדף, מחוללים. 4. חלופות ריכוז או ביזור (מע' מרכזית או מחשבים אזוריים או רשת, תחנות עבודה וכו'). 5. חלופות במנשק אנוש-מחשב. 6. חלופות מיכון מול מעורבות משתמש.</p>	חלופות
82	(תכן מונחה זרימת נתונים)	<p>כל המודולים הכפופים (מדרגית) למודול זה – כלומר, כל שינוי במודול ישפיע רק על הכפופים לו ולא במקומות אחרים במע'.</p>	טווח הבקרה של מודול
82	(תכן מונחה זרימת נתונים)	<p>כל המודולים המושפעים ע"י החלטה במודול זה.</p>	טווח השפעה של מודול
97	<p>< תכן מונחה עצמים < Link < Aggregation</p>	<p>היחסים בין 2 עצמים הם בעיקר מ-2 סוגים: 1. Link. 2. Aggregation.</p>	יחסים בין עצמים
8	< הנדסת תוכנה	<p>חשיבות ומאפייני בעיית הייצור – ראה עמ' 8. כולל טבלה עם הזמן הדרוש והעלות למס' פקודות בשפה עלית. מסקנה: יעילות גבוהה ואיכות טובה בייצור תוכנה הם גורמי המפתח להצלחה בתחרות ובהישרדות.</p>	ייצור
14	< מערכת תוכנה, שלבי פיתוח	<p>בשלב זה נקבע ה"איך".</p>	יישום
14	< מודל מפל המים	<p>תכן על ומפורט, קידוד ובדיקה.</p>	יישום
34, 33	<p>< COCOMO < פרויקטים, סוגים < Organic < Semidetached < Embedded < MM</p>	<p>עמ' 33 - טבלה הכוללת: 1. תכונות המוצר. 2. תכונות החומרה. 3. תכונות אנוש. 4. תכונות הפרוייקט. ובסופה נוסחה לחישוב מאמץ פיתוח נדרש לסוגי הפרוייקטים השונים וזמן הפיתוח.</p>	כופלי מאמץ
52	(מבנה מע' מחשב)	<p>תרשים</p>	כלי תוכנה, סיווג
76	<p>< תכן ארכיטקטורה < אי-תלות < פונקציונלית < צימוד</p>	<p>התמקדות כל מודול בנושא ספציפי אחד. הטווח הוא ממפוזר ועד "single minded" – התמקדות בפרטים ספציפיים – באופן ממוקד.</p>	ליכוד Cohesion
100	(תכן מונחה עצמים מחלקות ועצמים)	<p>מידת הקשר בין אלמנטים של מחלקה אחת או של עצם אחד.</p>	ליכוד Cohesion
130	(מושגי יסוד - אמינות)	<p>פגם אשר בתנאים מסוימים של הרצת התוכנה גורם לכישלון. ליקוי אחד יכול להיות הגורם למס' סוגי כישלונות.</p>	ליקוי בתוכנה
15	< אבטיפוס		לקוח
7	<p>< הנדסת מערכות < ממוחשבות < הקצאה < STP</p>	<p>מתחיל עם הגדרת מטרות ואילוצים ע"י הלקוח ויוצר הצגה מחדש של תפקוד, ביצועים, מנשקים, אילוצי עיצוב ותבניות מידע. כיוון שהיצירה של רוב המע' החדשות מתחילה עם מושג מעורפל של התפקוד המבוקש, מהנדס המע' חייב לתחום את המע' ע"י איתור תחום התפקוד והתצוגה המבוקשים. הוא המבצע את ה-STP.</p>	מהנדס מערכות מחשב = מנתח מערכות
90	< תכן מונחה עצמים	<p>חלק ניתן להפרדה לוגית של מערכת.</p>	מודול
75	(עקרונות תכן) תכן נתונים	<p>תוכנה מחולקת לאלמנטים נפרדים של פונקציות קשירות ובלתי תלויות שנקראים מודולים. לכל מודול שם וכתובת משלו. חשיבות החלוקה למודולים נובעת מהעובדה שהמורכבות הכוללת של בעיה גדולה</p>	מודולריות

		מסכום המורכבויות של מרכיביה, וכן מאמץ/זמן הפתרון של בעיה בכללותה גדול מסכום המאמצים/זמנים לפתרון מרכיביה. ראה משוואה עמ' 75.	
94	(עקרונות מודל העצמים) מודל העצמים <	תכונת מע' להיות מופרדת לקב' של מודולים מלוכדים המקושרים באופן רופף.	<u>מודלריות</u>
79	(ארכיטקטורת תכן) מודל Interrupt Driven מע' Event Driven <	מודל במערכות מונחות אירועים (Event Driven). מאורע משודר לכל תת-מע'. כל תת-מע' המסוגלת לטפל במאורע תעשה זאת. למשל: לוויינים שמשדרים נתוני מז"א: כל כמה שניות נשלחת תמונה מהלוויין וכל תחנת קרקע שרוצה תמונה מקבלת.	<u>מודל Broadcast</u>
79	(ארכיטקטורת תכן) מודל Broadcast מע' Event Driven <	מודל במערכות מונחות אירועים (Event Driven). משמש במע' זמן אמת בה פסיקות מזהות ע"י מנהל הפסיקות ומועברות לרכיב המתאים לטיפול בהם. למשל: חולה בטיפול נמרץ, מערכות מוטסות, בקרה אווירית (מטוס שנמצא בגובה/מסלול לא נכון, מרכזיית טלפונים).	<u>מודל Interrupt-Driven</u>
78, 77	ארכיטקטורת תכן ארכיטקטורת שרת לקוח מע' Event Driven <	תת-מערכות חייבות להעביר ביניהן נתונים. ניתן לבצע זאת באחת מ-2 הדרכים: 1. נתונים משותפים מוחזקים במאגר נתונים מרכזי ולכל תת-מע' קיימת גישה אליהם. 2. כל תת-מע' מחזיקה מאגר נתונים משלה ומעבירה נתונים נדרשים לתת-מע' האחרות. כאשר כמויות גדולות של נתונים הם משותפים, מוגדל המאגר הוא הנפוץ ביותר. יתרונות: 1. דרך יעילה לשיתוף כמויות גדולות של נתונים. 2. תת-מע' אינן צריכות לדאוג לאגירה ולניהול כולל של גיבויים, אבטחת מידע וכדו'. 3. מודל השיתוף ברור וקל להשתלב בו. חסרונות: 1. תת-מע' חייבות להסכים למודל המאגר ובאופן בלתי נמנע לתקבל פשרה. 2. התפתחות הנתונים קשה ויקרה – הוספת שדה/רשומה, שינוי רשומה: לא ניתן להכניס שינויים במאגר המרכזי עד אשר כל תת-מע' יתארגנו להתמודד עם השינוי. 3. אין מדיניות ניהול ספציפית: אין אף אחד שאחרי ישירות על המאגר כי כולם אחראים עליו. 4. קשה לבצע ביזור באופן יעיל - אם רוצים שיהיה עותק של ה-db בכמה מקומות: אם זה מאגר אחד גדול אזי כל עדכון שמתבצע צריך כל פעם לגשת לכמות גדולה של נתונים ולו זה היה מפוזר בכמה מקומות אזי ניתן היה לעשות במקום מסוים חתך על כמות נתונים קטנה יותר. 5. תת-מע' אחת צריכה להכיל פרטים שתת-מע' אחרת לא צריכה כלל ויש מקרים שיצטרכו לרוץ על כל השדות הללו עד שנגיע לנתונים הרלוונטיים.	<u>מודל המאגר</u>
62	(בניית מודל ניתוח) ניתוח, בניית מודל מודל פונקציונלי מודל התנהגותי ERD <	זיהוי והגדרת הנתונים והיחסים ביניהם. מקובל להשתמש ב-ERD.	<u>מודל הנתונים</u>
93, 94, 95, 97, 96	תכן מונחה עצמים הפשטה הסתרת מידע מודלריות הורשה/מדרג סיווג מקביליות התמדה Polymorphism מנשק המחלקה Reuse Association Message <	טכנולוגית העצמים בנויה על יסודות הנדסיים מבוססים ששילובם נקרא מודל Object-ה. אף אחד מהעקרונות האלה איננו חדש אך החשיבות של מודל העצמים פירוט העקרונות: 1. הפשטה. 2. הסתרת מידע. 3. מודלריות. 4. הורשה/מדרג. 5. סיווג. 6. מקביליות 7. התמדה 8. Polymorphism 9. Message (Communication) 10. Association 11. Reuse (מיחזור) 12. מנשק המחלקה (Class Interface).	<u>מודל העצמים</u> <u>מודל object</u>
63	(בניית מודל ניתוח) ניתוח, בניית מודל מודל פונקציונלי מודל הנתונים STD <	אם נדרש. תיאור מצבים ומאורעות הגורמים למעבר בין המצבים. מקובל להשתמש ב-STD.	<u>מודל התנהגותי</u>
36, 35	COCOMO מודל פוסט ארכיטקטוני רישום מוקדם <	מודל ב - COCOMO2. מציע טווח הערכה על העלות, המאמץ וטווח הזמנים, מהתוצאה הטובה ביותר ועד התוצאה הגרועה ביותר. מאפשר למתכנן לעשות תרחיש "מה אם..." ע"י הדגמה מהירה איך ישפיעו דרישות ההסתגלות, המקורות והדחיסה על העלויות ולוח הזמנים.	<u>מודל יישומי</u>
62	(בניית מודל ניתוח) ניתוח, בניית מודל <	תאור תפקוד המע' והתמרות (transform) הנתונים (למשל ע"י DFD).	<u>מודל פונקציונלי</u>

	DFD <		
35	COCOMO מודל יישומי רישום מוקדם		מודל ב - COCOMO2 <u>מודל פוסט ארכיטקטוני</u>
16			<u>מחוללי יישומים</u>
90	(תכן מונחה עצמים)		מושגים בתכן מונחה עצמים <u>מחלקה - Class</u>
100, 101	(תכן מונחה עצמים) Key Abstraction < צימוד < ליכוד < Sufficiency < Completeness < Primitiveness <		<u>מחלקות ועצמים</u> במהלך הניתוח ובשלב המוקדמים של התכן על המפתח: 1. לזהות מחלקות ועצמים המהווים את הלקסיקון של מרחב הבעיה = key abstraction של הבעיה. 2. לקבוע את המבנים שבאמצעותם קבוצות של עצמים מספקים את ההתנהגויות העונות על דרישות הבעיה. מבנים cooperative או נקראים mechanism היישום. השיקולים בבחירת המחלקות והעצמים הם: 1. צימוד. 2. ליכוד. 3. Sufficiency. 4. Completeness. 5. Primitiveness. השיקולים של צימוד וליכוד מועתקים מהתכן המבני בו הם מתייחסים למודולים. השיקולים של Sufficiency ו-Completeness מוודאים שכל המאפיינים המשמעותיים נכללים וכל המאפיינים הנכללים הם משמעותיים (שלא שכחנו להכליל שום דבר ומצד שני שמה שברחנו ותכונותיו משמעותיות, כלומר נדרש עבור היישום המסוים, כלומר: לא להעתיק כדי "שיהיה" כשנצטרך). באמצעות מדדים אלו ניתן למדוד את איכות ההפשטה.
105, 106	(שיטה לקביעת מחלקות ועצמים בתכן מונחה עצמים) מחלקות ועצמים, שיטות לקביעה <		<u>מחלקות ועצמים Use-Case</u> קובעים אלו תרחישים מתארים באופן בסיסי את תפקוד המע', מנתחים כל תסריט ומזהים בו את העצמים המשתתפים בתרחיש. בהמשך מרחיבים את התרחישים המקוריים כדי לטפל גם במקרים חריגים ואז מוספים או מעדכנים את העצמים במידת הצורך. גישה זו יכולה להיות משולבת בכל אחת מהגישות הקודמות.
105	(שיטה לקביעת מחלקות ועצמים בתכן מונחה עצמים) מחלקות ועצמים, שיטות לקביעה <		<u>מחלקות ועצמים השיטה התנהגותית</u> בעוד שבגישה הקלאסית מתמקדים בדברים מוחשיים עפ"י גישה זו מתמקדים בהתנהגות הדינמית כמקור עיקרי לקביעת מחלקות ועצמים. לפי Wirfs-Brock נקבעים העצמים עפ"י אחריות. האחריות של עצמים הם כל השירות שהעצם מספק. עצמים בעלי אחריות משותפת מאוגדים למחלקה ומחלקות כאלו ל-superclass. Rubin & Goldberg מציעים לזהות עצמים ומחלקות מתפקידי המע': תחילה מנסים להבין מה קורה במע' והיזמים והמשתתפים בעלי תפקידים ראשיים מזהים כעצמים. תפיסה זו קרובה לרעיון של function points כאשר FP היא פונקציה עסקית אחת של משתמש סופי, כגון: קלט, פלט, שאילתה וכו'.
104, 105	(שיטה לקביעת מחלקות ועצמים בתכן מונחה עצמים) מחלקות ועצמים, שיטות לקביעה <		<u>מחלקות ועצמים השיטה הקלאסית</u> נקראת כך בגלל דמיונה לגישת הסיווג הקלאסית. עפ"י Shlaer & Mellor: <ul style="list-style-type: none"> דברים מוחשיים, כגון: מכונית, ציונים, נתוני חום. תפקידים, כגון: מורה, אמה, פוליטיקאי. מאורעות, כגון: נחיתה, פסיקה, בקשה. פעולות גומלין, כגון: הלוואה, פגישה, חצייה. עפ"י Ross: <ul style="list-style-type: none"> אנשים: בנ"א המבצעים תפקיד מסוים. כלומר, אם רוצים לסווג אנשים אזי נחפש תפקידים של אנשים ולפי זה נסווג אותם. מקומות - אזורים המוקצים לאנשים או חפצים. חפצים - ישויות גשמיות או מוחשיות. ארגונים - אוסף מאורגן של אנשים, משאבים, מתקנים ויכולות בעלי מטרה מוגדרת. תפיסות - עקרונות או רעיונות שאינם מוחשיים המשמשים לפעולות עסקיות ו/או לתקשורת. מאורעות - דברים שקורים בד"כ במקום ובזמן מסוים או כצעדים באופן סדרתי. עפ"י Coad & Yourdon: <ul style="list-style-type: none"> מבנה: יחסי is-a או part-of. מע' אחרות: מע' חיצונית איתן היישום פועל. התקנים: התקנים איתם היישום פועל. מאורעות שזוכרים: מאורעות היסטוריים שצריכים להיות רשומים. תפקידים משוחקים: תפקידים שונים של המשתמשים בפעולה עם היישום. מיקומים: מיקומים פיזיים, משרדים ואתרים שחשובים ליישום. יחידות ארגוניות: קבוצות אליהן המשתמשים משתייכים.
105	(שיטה לקביעת מחלקות ועצמים בתכן מונחה עצמים) מחלקות ועצמים, שיטות לקביעה <		<u>מחלקות ועצמים השיטה התחומית</u> השיטות הקודמות מתייחסות לפיתוח מע' אחת ספציפית ואילו גישה זו מזהה עצמים ומחלקות המשותפות לכל סוגי היישומים בתחום מסוים. ניתן להתייחס ליישומים דומים, vertical domain analysis (אנכי), או לחלקים דומים של אותו יישום, horizontal domain analysis (אופקי). בפרט במע' חדשה ניתן להיעזר בקיימת.

		השיטה היא להסתמך על מומחי תחום שמכירים היטב את כל המרכיבים של בעיה מסוימת. מומחה כזה יכול להיות משתמש סופי של המע', כגון: רופא.	
106	<p>(שיטה לקביעת מחלקות ועצמים בתכנ מונחה עצמים) <</p> <p>מחלקות ועצמים, שיטות לקביעה <</p>	<p>טכניקה המשמשת בעיקר ל-use case analysis ובה מאתרים מחלקות בתרחישים. כל מחלקה שזוהתה נרשמת על כרטיס כאשר בראש הכרטיס רושמים את שם המחלקה, בחציו האחד את האחראיות שלו ובחציו האחר את משתפי הפעולה שלו. תוך כדי המעבר על התרחיש ניתן להוסיף אחראיות נוספות וכתוצאה מכך ייתכן 2-כרטיסים יאוחדו לכרטיס אחד או שכרטיס יפוצל ל-2 כרטיסים.</p>	<p><u>מחלקות ועצמים, כרטיסי CRC</u></p>
106	<p>(שיטה לקביעת מחלקות ועצמים בתכנ מונחה עצמים) <</p> <p>מחלקות ועצמים, שיטות לקביעה <</p>	<p>זיהוי המחלקות והעצמים מתוך ניתוח מבני של המע'. היתרון (הכמעט יחיד) של גישה זו הוא להסתמך על ניסיונם של מנתחי מע' שעד כה הי בעיקרו במתודולוגיה של ניתוח מבני. הניתוח המבני ישמש רק כ-front end לניתוח ה-OO.</p>	<p><u>מחלקות ועצמים, ניתוח מבני</u></p>
104	<p>השיטה הקלאסית <</p> <p>השיטה <</p> <p>ההתנהגותית <</p> <p>השיטה התחומית <</p> <p>Use Case <</p> <p>כרטיסי CRC <</p> <p>תיאור מילולי לא <</p> <p>פורמלי <</p> <p>ניתוח מבני <</p>	<p>1. השיטה הקלאסית. 2. השיטה ההתנהגותית. 3. השיטה התחומית. 4. Use Case. 5. כרטיסי CRC. 6. תיאור מילולי לא פורמלי. 7. ניתוח מבני. 8. Key Abstractions and Mechanisms</p>	<p><u>מחלקות ועצמים, שיטות לקביעה</u></p>
106	<p>(שיטה לקביעת מחלקות ועצמים בתכנ מונחה עצמים) <</p> <p>מחלקות ועצמים, שיטות לקביעה <</p>	<p>רושמים את תיאור הבעיה בשפה טבעית, אח"כ מדגישים את כל שמות העצם ואת הפעלים בקו תחתון. שמות העצם יהיו מועמדים לעצמים והפעלים יהיו הפעולות שלהם. גישה זאת הוצעה ע"י Abbott והיא פשוטה ומתאימה לבעיות טריוויאליות. שפה טבעית היא לא מדויקת והתוצאה של הניתוח תהיה תלויה במומחיות הכתיבה של הכותב.</p>	<p><u>מחלקות ועצמים, תיאור מילולי לא פורמלי</u></p>
99	<p>(תכן מונחה עצמים) <</p> <p>Generalization <</p> <p>Whole/Part <</p> <p>Association <</p> <p>Inheritance <</p> <p>Aggregation <</p> <p>Using <</p> <p>Instantiation <</p> <p>Metaclass <</p>	<p>יחסים בין מחלקות הם בעיקרם מאחד מ-3 סוגים: 1. generalization/specialization. 2. whole/part. 3. association. סוגים אלו מתחלקים ל-6 היחסים הבאים: 1. Association. 2. Inheritance. 3. Aggregation. 4. Using. 5. Instantiation. 6. Metaclass.</p>	<p><u>מחלקות, יחסים בין</u></p>
123	<p>(הבטחת איכות תוכנה)</p>	<p>כך מכונה פעילות צוות ה-SQA, פעילות המעורבת בכל שלב של תהליך הנדסת התוכנה. מטרתה העיקרית היא לאתר ולהסיר פגמים בייצור התוכנה, כאשר עדיין זול למצוא ולתקן אותם.</p>	<p><u>מטריה</u></p>
11	<p>משבר התוכנה <</p>	<p>של מנהלים, לקוחות, מקצועני תוכנה. גורמים להצלחה ולכישלון של פרוייקטים. מיתוסים על שיטות מפרט פורמליות.</p>	<p><u>מיתוסים</u></p>
138	<p>פרוייקטים <</p>	<p>אחרים לתכנון וללוח הזמנים של פיתוח הפרוייקט. מפקחים על העבודה עמ"נ להבטיח שהעבודה מתבצעת עפ"י הסטנדרטים הדרושים. עוקבים אחרי התקדמות הפרוייקט כדי להבטיח שלא תהיה חריגה בתקציב ועמידה בזמנים. ניהול טוב לא יכול להבטיח את הצלחת הפרוייקט אולם ניהול גרוע לרוב גורם לכישלונות.</p>	<p><u>מנהלי פרוייקט</u></p>
15, 54, 53	<p>(חלקי מע' תוכנה) <</p> <p>ממשק משתמש <</p>	<p>Human Computer Interface – HCI</p>	<p><u>ממשק אנוש מחשב</u></p>
97	<p>(תכן מונחה עצמים) <</p> <p>מודל העצמים. <</p> <p>Public <</p> <p>Protected <</p> <p>Private <</p>	<p>הממשק של מחלקה נותן את המבט החיצוני ולפיכך מדגיש את ההפשטה תוך הסתרת המבנה וההתנהגות שלו. הממשק מתחלק ל-3 חלקים: 1. Public. 2. Protected. 3. Private.</p>	<p><u>ממשק מחלקה</u></p>
74, 110, 111	<p>אבטיפוס <</p> <p>צבעים <</p> <p>ממשק אנוש-מחשב <</p>	<p>תכן ממשק המשתמש חייב לקחת בחשבון את הדרישות, הניסיון והיכולות של משתמשי המע'. המשתמשים צריכים להיות מעורבים בתהליך התכן והתכן צריך להיות מוצג ולהתעדכן ע"י prototyping. קיימים גורמים הכרטיים שהמתכנן חייב להיות מודע להם, כגון: קיבולת זיכרון קצר מועד. <ul style="list-style-type: none"> הממשק צריך להיות מבוסס על מושגים ומונחים מונחי-משתמש ולא על מושגי מחשב. המע' חייבת להציג דרגה גבוהה של עקביות בשימוש. אסור למע' להפתיע את המשתמש. פקודות דומות חייבות להתנהג בצורה דומה. המע' חייבת לספק מידה מסוימת של סובלנות לשגיאות משתמש ולאפשר למשתמש להתאושש משגיאות (למשל: undo, אישור למחיקה וכו'). חייבים לספק הדרכה מקוונת למשתמש (למשל: help, online manual). יש לאפשר התאמה אישית. יש לעדכן את המשתמש באופן שוטף בפעולות המע'. </p>	<p><u>ממשק משתמש (UI), עקרונות תכן</u></p>

		<ul style="list-style-type: none"> יש להשתמש בשפה פשוטה וברורה, ללא קיצורים. יש לשאוף למע' חסונה שהיא מצד אחד חסינה לשגיאות ומצד שני מספקת ברירות מחדל לנתונים חסרים ומתקנת שגיאות ברורות. יש להתחשב באסתטיקה חזותית. יש לנצל תכונות של התקנים מודרניים (גרפיקה, צבעים, קול וכיו"ב). יש לפשר למשתמש לבטל את הקול, למעט הודעות שגיאה. עיקרון KISS (Keep It Simple Stupid). שימוש בצבעים עפ"י B. Shneiderman - ראה צבעים. 	
57, 58, 59, 61, 60	(ניתוח דרישות תוכנה)	<p>תכנון הניתוח (שאלות) ותוצרים. שאלות עליהן צריך מנתח המע' לענות בנושא: אנשי קשר של המשתמש; מטרות המע'; המע' הנוכחית; מידע ומבני נתונים; משתמשים; מחקר על מע' אחרות; הצעת חלופות; ניתוח מבנה התוכנה; תוכניות לשלב הבא; Reviews.</p>	<u>מנתח מערכות, רשימת תיג</u>
55, 43		מפרטי הנדסת מערכת, כולל את SSS וכן OCD ו- SSDD	<u>מסמך ייזום</u>
79, 78	<ul style="list-style-type: none"> ארכיטקטורת תכן מודל המאגר ארכיטקטורת שרת לקוח מודל Broadcast מודל Interrupt Driven 	<p>מע' המונעות ע"י מאורעות חיצוניים, בהן תזמון המאורע הוא מחוץ לבקרה של תת-המע' המעבדת את המאורע. תת-המע' לא שולטת מתי קורים האירועים. המע' רק יודעת שכאשר קורים האירועים אזי צריכה להיות איזושהי תגובה (למשל: מתי יורד לחולה בטיפול נמרץ לחץ הדם).</p> <p>2 הסוגים העיקריים של מודלים מונעי מאורע הם:</p> <ol style="list-style-type: none"> 1. מודל Broadcast. 2. מודל Interrupt-Driven. 	<u>מערכות מונעות ע"י מאורעות / מערכות תגובתיות Event Driven / Reactive</u>
20	<ul style="list-style-type: none"> Reengineering Cobol Fortran 	מערכות ישנות שעדיין זקוקות לתחזוקה. כמות הקודים בהם עצומה. רוב מערכות אלו נכתבו ב-COBOL או FORTRAN.	<u>מערכות מורשה (Legacy Systems)</u>
13, 44, 14	<ul style="list-style-type: none"> מודל מפל המים אבטיפוס 4GT המודל הספירלי Reengineering 	<p>פיתוח מע' תוכנה מורכב מ-3 חלקים עיקריים:</p> <ol style="list-style-type: none"> 1. הגדרה – בשלב זה נקבע ה"מה". 2. יישום – בשלב זה נקבע ה"איך". 3. תחזוקה. <p>המערכת מורכבת מ-4 חלקים:</p> <ol style="list-style-type: none"> 1. חומרה. 2. תוכנה. 3. קושחה: צריבת האלגוריתמים על מעגלים / לוגיקה: הפיכת תוכנה לחומרה. 4. אונשה: האנשים במערכת. 	<u>מערכות תוכנה, שלבי פיתוח</u>
37, 38, 39, 42, 112	<ul style="list-style-type: none"> 498 סקר Functional Baseline Allocated Baseline Top Level Design 	<p>US-DOD-MIL-STD-498. ה-498 ירש את התקן 2167A. ה-498 מסתיים בסוף פיתוח הפרוייקט.</p> <p><u>ראה תרשימים – עמ' 37, 39.</u> <u>אחרי שלב ניתוח המערכות:</u></p> <p>הנדסת המערכת (מסמך ייזום): OCD, SSDD, SSS. אחרי ה-SSS ניתן לפתח 2 מסמכים שהם בין שלב הנדסת המע' לשלב פיתוח התוכנה: SDP, STP. שלב ניתוח דרישות התוכנה (תיק אפיון): SRS, IRS, HRS. בשלב זה מחלקים את המע' ל-CSC. אח"כ שלב התכן (תיק עיצוב): SDD, DBDD, IDD. בתום שלב זה כותבים את ה-STD וכן ניתן לכתוב גם את ה-STrP וה-SIP. קידוד-תכנות: SPS. בשלב זה ניתן גם לכתוב את אוסף המדריכים, או שכותבים אותם כשסתמים השלב: CPM, SIOM, COM, SCOM, SUM, FSM, SVD, STR. כל המסמכים יכולים לחזור על עצמם מס' פעמים כנדרש במע'. סקרים: ה-SDR, SRR, SSR, PDR, CDR, TRR, FQR היו חובה ב-2167A ואינם חובה ב-498. סקרי דרישות מערכת: SDR, SRR. סקרי תכן: PDR, CDR. לפי Pressman ה-SDR וה-SRR משולבים לסקר אחד: System Definition (Review).</p>	<u>מערכות תוכנה, תקנים לפיתוח</u>
50	<ul style="list-style-type: none"> חומרה תוכנה ומסד נתונים אנשים תיעוד נהלים 	חומרה, תוכנה ומסד נתונים, אנשים, תיעוד כולל (נהלים/הוראות). תרשים – עמ' 50.	<u>מערכת המחשב, חלקי</u>
		ראה מערכת תוכנה, שלבי פיתוח.	<u>מערכת תוכנה, מחזור חיים</u>
14	<ul style="list-style-type: none"> הנדסת מערכת ניתוח יישום מערכות תוכנה, שלבי פיתוח 	מודל של מחזור חיים של מע' תוכנה. במודל זה הלקוח אינו מתערב עד שהוא מקבל את המוצר המוגמר. תרשים – עמ' 14.	<u>מפל המים, מודל</u>
95	(עקרונות מודל העצמים) מודל העצמים	התכונה המבחינה בין עצם שהוא active לבין עצם שאינו active. במע' המבוססת על OOD ניתן להתייחס לעולם כמורכב מאוסף עצמים קואופרטיביים שחלקם	<u>מקביליות</u>

		active ולכן משמשים כמרכזי activity בלתי תלויים. ברגע שקיימת מקבילות במע' חייבים לקחת בחשבון כיצד עצמים שהם Active מסנכרנים את הפעילויות שלהם זה עם זה וכן עם עצמים שהם לגמרי סדרתיים. מימוש של מקבילות יכול להיות תכונה של שפת התכנות או ע"י שימוש בתוכניות ספריה מיוחדות לכך או ע"י שימוש בפסיקות הנותן אשליה של בו-זמניות.	
10, 9	מיתוסים	מחיר איכות לא מספקת – דוגמאות.	משבר התוכנה
64	ניתוח, בניית מודל	נחלקות ל-4 קבוצות עיקריות: 1. Data Flow Oriented – שיטות מונחות זרימת נתונים. 2. Data Structure Oriented – שיטות מונחות מבנה נתונים. 3. Object Oriented – שיטות מונחות עצמים. 4. Language-based Formal Specification – מפרטים מבוססי שפה פורמלית. לכל קבוצה פותחו אמצעי עזר ממוחשבים.	מתודולוגיות הכנת מפרט דרישות תוכנה
43	מערכות תוכנה, תקנים לפיתוח	מבנה מטריציאונלי. מלווה את המע' בכל זמן חייה, גם לאחר הסיום. לתיק התחזוקה בנוהל מפת"ח אין מקבילה ב-498.	נוהל מפת"ח
12	RMMM	תפקידם של מנהלי הפרוייקט הוא להבטיח שפיתוח התוכנה תואם את מדיניות הארגון, מטרותיו ודרישותיו.	ניהול פרוייקט
14	מודל מפל המים	הגדרת דרישות התוכנה.	ניתוח
17	המודל הספירלי	ניתוח חלופות וזיהוי/צמצום סיכונים.	ניתוח סיכונים
63, 62	עקרונות מפרט דרישות תוכנה מודל פונקציונלי מודל נתונים מודל התנהגותי מתודולוגית הכנת מפרט דרישות תוכנה	מודל הניתוח מורכב מ: 1. המודל הפונקציונלי. 2. מודל הנתונים. 3. המודל ההתנהגותי (אם נדרש) תרשים – עמ' 63.	ניתוח, בניית מודל
127, 128	(מדידת איכות תוכנה)	Complexity Measure.	סיבוכיות, מדד
128		ראה McCabe's Complexity Measure.	סיבוכיות ציקלומטית
95, 94	(עקרונות מודל העצמים) מודל העצמים Static/Early Binding Late/Dynamic Binding	האכיפה של מחלקה, של עצם המונע מעצמים מסוגים שונים להתחלף או לכל היותר מאשר להם להתחלף באופן מאוד מוגבל. הסיווג מאפשר להביע את הפשטות באופן ששפת התכנות בה ממשים אותה אוכפת את החלטות התכנן.	סיווג
90	(תכן מונחה עצמים)	מושגים בתכן מונחה עצמים.	סיווג – Typing
101, 102, 103, 104	סיווג קלאסי אישכול תפיסתי תיאורית אבטיפוס	הזיהוי דורש יכולת גילוי והמצאה. הגילוי מזוהה עם ניתוח וההמצאה עם תכן. 2 מתכנתים יכולים לקבוע סיווגים שונים – ושניהם יהיו נכונים. אין מבנה מחלקה מושלם ואין קבוצה נכונה של עצמים. הבחירה היא בסופו של דבר פשרה המעוצבת ע"י שיקולים מתחרים רבים. סיווג הוא באופן בסיס בעיה של clustering. התהליך הוא מדורג ואיטרטיבי והקושי שלו נובע בעיקר מכך שקבוצה נתונה של עצמים ניתנת לסיווג במס' רב של אופנים נכונים. 3 הגישות לסיווג הן: 1. סיווג קלאסי. 2. אישכול תפיסתי. 3. תיאוריית אבטיפוס. מניסיון, כדאי לבצע קודם סיווג קלאסי, אם זה לא מצליח אז אישכול תפיסתי ואם גם זה לא מצליח אז עפ"י הדמיון לאבטיפוס.	סיווג מחלקות ועצמים
103	(תכן מונחה עצמים) סיווג מחלקות ועצמים	סיווג עפ"י תכונות. כל הישויות בעלות תכונה או אוסף תכונות משותף מהוות קטגוריה אחת. למשל: "אנשים נשואים" מהווים קטגוריה. לעומת זאת "אנשים גבוהים" אינם קטגוריה אלא אם כן יהיה מוסכם מאיזה גובה אדם נקרא גבוה. תכונה יכולה להיות בעלת חשיבות ביישום אחד, בעוד ביישום אחר היא חסרת חשיבות לחלוטין. לכן אין שום מדדים מוחלטים לסיווג.	סיווג קלאסי
28	סיכונים, ניהול	1. Catastrophic: כשלון המשימה ו/או תוספת עלות גדול מ-\$500K. 2. Critical: קרוב לכישלון ו/או תוספת עלות \$100K-\$500K. 3. Marginal: ירידה בביצועים ו/או תוספת עלות \$1K-\$100K. 4. Negligible: אי נוחות ו/או תוספת עלות זניחים (קטן מ-\$1K).	סיכון, השפעות
27	סיכונים, ניהול	1. סיכון ביצועים: המע' לא תבצע את מה שהיא צריכה לבצע ב-performance- שהיא צריכה. 2. סיכון עלות: הפרוייקט לא יסתיים בעלות הנדרשת. 3. סיכון סיוע: סיכונים בעזרים המשמשים לפרוייקט (למשל: אין debugger ל-compiler). 4. סיכון לר"ז.	סיכון, מרכיבים
28	סיכונים, ניהול	לכל סיכון נקבע: 1. פירוט הסיכון. 2. סוג הסיכון: לפי סיכונים, ניהול. 3. הסתברות להיקרות הסיכון.	סיכונים, טבלה

		4. השפעתו: לפי סיכונים, השפעות. דוגמה.	
21	<ul style="list-style-type: none"> סיכון, מרכיבים < סיכון, השפעות < סיכון, טבלה < 	<p>סוגי סיכונים:</p> <ol style="list-style-type: none"> 1. גודל המערכת. 2. השפעות עסקיות, כגון אילוצי שוק (למשל: תוכנה עבור בחירות, מוצר עבור עונה מסוימת). 3. מאפייני המשתמשים (סיכון קטן עבור משתמשים שמבינים במחשבים; אפיון אחר עבור אנשים מבוגרים). 4. הגדרת תהליך פיתוח: איך מתבצע תהליך הפיתוח (עפ"י נהלים או עפ"י אינטואיציה – סיכון גדול). 5. סביבת הפיתוח: האם מסתמכים על חומרה מסוימת ובסופו של דבר משתמשים בחומרה אחרת? 6. טכנולוגיה מפותחת. 7. גודל צוות וניסיונו: צוות קטן עם אנשים מנוסים – סיכון קטן; צוות גדול וכמה לא מנוסים – סיכון להצלחה. 	<p>סיכונים, ניהול</p> <p>סיכונים, סוגים</p>
17, 19, 18	<ul style="list-style-type: none"> תכנון < ניתוח סיכונים < הנדסה < הערכת הלקוח < 	<p>פוחח ע"י Boehm כדי לשלב את התכונות הטובות ביותר ממודל מפל המים הקלאסי ומודל האבטיפוס תוך הוספת אלמנט ניתוח סיכונים.</p> <p>המודל מורכב מ-4 פעולות עיקריות:</p> <ol style="list-style-type: none"> 1. תכנון (Planning). 2. ניתוח סיכונים (Risk Analysis). 3. הנדסה (Engineering). 4. הערכת הלקוח (Customer Evaluation). <p>לכל חלק של המוצר ולכל רמת פירוט של פיתוח המוצר משתמשים באותה סדרת צעדים ובכל איטרציה סביב הספירל מהמרכז והחוצה מתקדמים בבניית גרסאות מושלמות יותר.</p> <p>יתרונות וחסרונות – עמ' 19.</p> <p>איור + הסברים – עמ' 18, 19.</p> <p>לפרוייקט תוכנה שמרכיביו מפותחים ע"י גופים או ארגונים שונים וכן לפרוייקטים המפותחים באופן תוספתי (Incremental) ניתן להשתמש בסדרה של ספירלים.</p>	<p>ספירלי, מודל</p>
39		<p>הלקוח/המפתח דנים בשלב שהסתיים.</p> <p>בסוף שלב ה-coding אין סקר ומבצעים אינטגרציה של CSC (מודלים ראשיים).</p>	<p>סקר</p>
		<p>ראה SSR.</p>	<p>סקר מפרט תוכנה</p>
117	<ul style="list-style-type: none"> (בדיקות תוכנה) < בדיקות פנימיות < 	<p>סקרים פנימיים, פורמליים, שמטרתם לאשר סיום יחידת פיתוח. בד"כ מבצעים אותם לפני ביצוע בדיקות חיצוניות.</p>	<p>סקרי סיכום</p>
36	<ul style="list-style-type: none"> (חישוב עלות פרוייקט) < 	<p>רשימה.</p>	<p>עלויות פרוייקט, עצות ייעול</p>
90, 91, 93, 92	<ul style="list-style-type: none"> תכן מונחה עצמים < מודולריות < הפשטה < הסתרת מידע < מודל העצמים < 	<p>רכיב/יישות של העולם האמיתי ממופה לתחום התוכנה.</p> <p>מיפוי העצם מגדיר:</p> <ol style="list-style-type: none"> 1. מבנה נתונים. 2. תהליכי טיפול במבנה הנתונים (פעולות). 3. הודעות לביצוע התהליכים על המבנה. <p>החלק הפרטי של העצם הוא מבנה הנתונים ותהליכי הטיפול בו.</p> <p>החלק המשותף של עצם הם ההודעות המהוות את הממשק אליו. הודעות אלו אומרות לו "מה" לבצע, אך לא "איך".</p> <p>עצמים מאורגנים בצורה מדרגית. כל עצם שייך למחלקה מסוימת ויורש את מבנה הנתונים הפרטי מהמחלקה. לפיכך, עצם הוא הדגם של מחלקה שלכל חברה תכונות דומות.</p> <p>תיאור עצם נעשה ב-2 חלקים:</p> <ol style="list-style-type: none"> 1. תיאור פרוטוקול בו מתוארות ההודעות שהעצם מקבל והפעולות שהוא מבצע עם קבלתן ("מה"). 2. תיאור מימוש בו מתואר במפורט כיצד העצם מבצע את הפעולות עפ"י ההודעות ("איך"). זה כולל את המתבצע בחלק הפרטי של העצם (תיאור הפרוטוקול מתאים לרמת-על של התכן ותיאור היישום למפורט). <p>למשל: עצם שיועד לחשב משוואות ריבועיות:</p> <p><i>חלק פרטי</i>: נוסחת חישוב, טיפול בשורשים.</p> <p><i>ה"מה"</i>: הפרמטרים שמקבל – לפחות 3 פרמטרים.</p> <p>דוגמאות – עמ' 91-93.</p>	<p>עצם – Object</p>
62		<p>רשימה – ראה עמ' 62.</p>	<p>עקרונות מפרט דרישות תוכנה</p>
45, 44	<ul style="list-style-type: none"> Build < Grand Design < Incremental < Evolutionary < Reengineering < 	<ol style="list-style-type: none"> 1. Grand Design. 2. Incremental. 3. Evolutionary. 4. Reengineering. 	<p>פיתוח מערכות, אסטרטגיות</p>
11, 8, 12	<ul style="list-style-type: none"> Boehm < 	<p>סיווג פרוייקטים לפי Boehm:</p> <p>קטן: עד 8,000 פקודות.</p> <p>בינוני: 8,000-32,000</p>	<p>פרוייקטים</p> <p>פיתוח פרוייקט</p>

			גדול: 128,000-32,000 גדול מאוד: 512,000-128,000 ענק: מעל 512,000 פקודות. גורמים הצלחה ולכישלון + סוגי כישלון – עמ' 11-12.
32	Organic Semidetached Embedded	<	1. Organic. 2. Semidetached. 3. Embedded.
			פשוטות
111	מנשק משתמש, עקרונות תכן	<	ראה Primitiveness. 1. אל תשתמש בצבעים רבים מדי, לא יותר מ-4-5 בו זמנית ובסה"כ לא יותר מ-7. 2. השתמש בקידוד צבעים כדי לסייע למשתמש לבצע משימותיו. 3. הרשה למשתמש לבקר קידוד הצבעים. 4. השתמש בקידוד הצבעים באופן עקבי. 5. השתמש בשינוי צבע להבלטת שינויי מצב. 6. המנע מזוגות צבעים המתנגשים. למשל: אדום וכחול. 7. יש לזכור שיש 6% מהאוכלוסייה שהם עיוורי צבעים (לא מסוגלים להבחין בין ירוק ואדום יחד).
76	תכן ארכיטקטורה אי-תלות פונקציונלית ליכוד	<	יחסי גומלין בין המודולים. באופן אידיאלי יש עצמאות מוחלטת. הטווח הוא מעצמאות מוחלטת ועד צימוד של מידע או בקרה על זרימת המידע (רצוי שיהיה מינימלי): מודול אחד משפיע על בקרת זרימת המידע של המודול השני.
100	מחלקות ועצמים	<	מידת העוצמה של הקשר בין השירותים.
97,98,99			ראה Aggregate
29			ראה RMMM.
14			
84,113,114,115	CDR תרשים מדרגי/מבני	<	תרגום הצגה מחדש של תוכנה באופן כזה שתוכל להיות "מובנת" ע"י המחשב. תהליך הקידוד הוא תהליך ההסבה לשפת תכנות. מלבד קידוד, מפתחים ברמות שונות של מערכות ניהול מידע יכולים עכשיו לתאר תוצאות רצויות, מלבד הליכים רצויים, בשפה לא נוהלית. אז ייווצר קוד מקור של שפת תכנות קונבנציונלית באופן אוטו'. הקידוד נחשב כצעד בתהליך הנדסת התוכנה. קידוד נחשב כתוצאה טבעית של תכנון. למרות זאת, מאפייני שפת תכנות וסגנון קידוד יכולים להשפיע עמוקות על איכות התוכנה ויכולת התחזוקה שלה. שלב הקידוד מתרגם תכן מפורט של תוכנה להבנת שפת תכנות. תהליך התרגום ממשיך כאשר המהדר מקבל קוד מקור כקלט ויוצר קוד יעד תלוי-מכונה כפלט. שלב התרגום ההתחלתי, מתכונן מפורט לשפת תכנון, הוא דאגה ראשונית בהקשר להנדסת התוכנה. "רעש" יכול לחדור לתהליך התרגום בדרכים רבות. פענוח לא נכון של תיאור תכנון מפורט יכול להוביל לקוד מקור שגוי. מורכבות או סייגי שפת תכנות יכולים להוביל לקוד מקור מפותל שהוא קשה לבחינה ולתחזוק. מאפיינים של שפת תכנות יכולים להשפיע על אופן חשיבתנו. התפשטות תכן מפורט לא בהכרח מגביל עיצובי תוכנה ותבניות נתונים. בשלב זה גם נכתבים המדריכים וה-SPS.
35	COCOMO מודל פוסט ארכיטקטוני מודל יישומי	<	מודל ב - COCOMO2.
104		<	ראה מחלקות ועצמים, שיטות לקביעה.
71,136,137,138,143,144,145,146	Data Flow Oriented Data Structure Oriented מתודולוגית הכנת מפרט דרישות תוכנה אבטיפוס Algebraic Specification Model-Based Specification	<	שיטות מפרט פורמליות נועדו לאפשר למהנדס תוכנה לתאר מפרט, לפתח ולאמת מע' תוכנה ע"י שימוש בסימון מתמטי קפדני. ע"כ ניתן לגלות רב-משמעות, חוסר שלמות וחוסר עקביות בשלבים מוקדמים של הפרוייקט ע"י שימוש בניתוח מתמטי (ולא ע"י סקרים). השיטות הפורמליות משמשות גם כבסיס לאימות התוכניות ומסייעות לגילוי שגיאות. השיטות הפורמליות מתחלקות ל: 1. Algebraic Specification. 2. Model-Based Specification. חסרונות: • עלות השימוש בשיטות אלו גבוהה יותר מאשר בשיטות לא פורמליות, בעיקר בשלב קביעת המפרט (הניתוח). תרשים – עמ' 136. • רוב שפות המפרט הפורמליות אינן תומכות ב-Rapid Prototyping המבוסס על מנשקים גרפיים הידודיים. • איכות מוצרי התוכנה משתפרת עם הזמן, גם ללא שיטות פורמליות. יתרונות: • במע' בהן התלות היא קריטית ובהן נדרשת בטיחות, אמינות ובטחון ברמה גבוהה השיטות הפורמליות נותנות אמון שמהערכות עונות על הדרישות. • סביר שהשיטות הפורמליות ישמשו להגדרת תקנים במידה רבה.

		בעד וגד + מסקנות – עמ' 142-146. מיתוסים בנוגע לשיטות מפרט פורמליות – עמ' 138.	
62		עקרונות מפרט דרישות תוכנה.	שיטות ניתוח
69		Data Structure Oriented. ראה Application Content.	תוכן היישום
100		ראה Completeness.	שלמות
52, 51	מערכת המחשב, חלקי	בנוסף על הדרישה של יכולת לפיתוח וביצוע המשימות, יש חשיבות רבה לשיקולים נוספים, כגון: 1. עלות. 2. בגרות התוכנה: למשל, האם כדאי להשתמש ב-win2000 שרק יצא – ניתן להגיד שאם הפרוייקט צריך להסתיים תוך 4 שנים אז כדאי. 3. ניסיון המפתחים או יכולת הלימוד. 4. תחזוקת התוכנה. 5. שילוב בקיים. תרשימים – עמ' 52.	תוכנה ומסד נתונים
13	תוכנה, תהליך עיבוד	אישור שמבטיח את התאימות לדרישות הלקוח.	תוכנה, אישור
116	בדיקות פנימיות בדיקות חיצוניות	הבדיקה מתחילה מבדיקה פנימית, החל מתת-תוכנית, פונקציה, דרך תוכנית ומודול ועד לשילוב בין המודולים.	תוכנה, בדיקות
13	תהליך עיבוד התוכנה	הגדרתם של תפקודי התוכנה והאילוצים המשפיעים על פעולתה.	תוכנה, הגדרת
13	תהליך עיבוד התוכנה	התוכנה חייבת להיות מפותחת לרמה שניתן יהיה לשנותה עפ"י צרכי הלקוח.	תוכנה, התפתחות
54, 57, 56	מנתח מערכות, רשימת תיג	הבנה מלאה של דרישות התוכנה חיונית להצלחה במאמצי פיתוח התוכנה. ניתוח הדרישות הוא תהליך של גילוי, שיפור, יצירת מודלים ופירוט. גם המפתח וגם הלקוח לוקחים חלק פעיל בדרכי ניתוח והגדרות. הלקוח מנסה לנסח מחדש מושג מעורפל בד"כ של תפקודי וביצועי תוכנה לכדי פרטים קונקרטיים. המפתח פועל כמתווך, כיועץ וכפותר בעיות. משימה של הנדסת תוכנה המגשרת על הפער שבין רמה מערכתית של הקצאות תוכנה לבין עיצוב תוכנה. דרישות ניתוח מקנות למעצב התוכנה הצגה של מידע ותפקוד שיכולים להיות מתורגמים לנתונים, לעיצוב ארכיטקטוני ולהליכי עיצוב. דרישות ההגדרות מקנות למפתח התוכנה וללקוח את האמצעים להעריך את איכות התוכנה בעת סיום בנייתה.	תוכנה, ניתוח דרישות
13			תוכנה, עיבוד
13			תוכנה, עלות התאמה
12, 7, 14, 13	תוכנה, תהליך עיבוד	ייצור המערכת עפ"י הדרישות והמפרטים שהוגדרו.	תוכנה, פיתוח
13	מחזור חיים של מע' תוכנה הגדרת תוכנה פיתוח תוכנה אישור תוכנה התפתחות התוכנה	מבוצע בד"כ ע"י מהנדסי התוכנה. סדרה של פעולות ותוצאות נלוות שאחראיות לייצורו של מוצר התוכנה. יש 4 עקרונות בסיסיים בפעולת העיבוד (משותף לכל עיבוד תוכנה): הגדרת תוכנה, פיתוח תוכנה, אישור התוכנה, התפתחות התוכנה. העלויות של התאמת התוכנה והאפשרות לשנותה עפ"י דרישות הלקוח היא בד"כ 60% מחירה הכולל של התוכנה. אחוז זה עולה ככל שמש' גדול יותר של תוכנות מיוצר וזקוק לעדכון ולאחזקה. לפיכך תכנון תוכנה גמישה הניתנת לשינוי הוא חיוני.	תוכנה, תהליך עיבוד
120	הנדסת תוכנה בקרת תצורה CM Corrective Maintenance Adaptive Maintenance Perfective Maintenance Preventive Maintenance	תחזוקה תוכנה מתחילה כבר בשלבי הפיתוח שלה ולא מסתיימת לעולם, זאת בגלל שתוכנה תמיד משתנה, יש bugs לתקן, להוסיף הרחבות, לבצע אופטימיזציות ולעשות שינויים. התחזוקה מחולקת ל-4 סוגים: 1. Corrective Maintenance. 2. Adaptive Maintenance. 3. Perfective Maintenance. 4. Preventive Maintenance. בפועל, החלוקה בין סוגי התחזוקה: 65% perfective, 18% adaptive, 17% corrective, Preventive & others – זניח: או שמראש כותבים את התוכנה כך שתהיה נוחה לתחזוקה או שלא מייחסים לזה חשיבות. נאי הכרחי לתחזוקה יעילה של תוכנה הוא פיתוחה עפ"י עקרונות הנדסת תוכנה. הבעייתיות הגדולה כיום בתחזוקה היא בעיקר בגלל פיתוח מערכות ללא התחשבות כלל בצורך תחזוקה, ללא תיעוד מספק ועם כוח אדם שכבר עזב את הארגון.	תוכנה, תחזוקה ובקרת תצורה
35		את התזמון המתוכנן כדאי להציג ע"י Gantt Chart ו/או PERT.	תזמון, הצגה
7, 14, 11			תחזוקה
9, 8		צמצום עלות תחזוקה	תחזוקה
20	מערכות מורשה	הוערך ש-80% מכלל הוצאות הכספיות של התוכנה הולכים לתחזוקת המע' והתפתחותה (Yourdon). מע' ישנות שעדיין זקוקות לתחזוקה נקראות מע' מורשה.	תחזוקה
103, 104	(תכן מונחה עצמים) סיווג מחלקות	סיווג עפ"י קשר עם אבטיפוס. הסיווג הקלאסי והאישיכול התפיסתי בד"כ מספיקים לבצע כמעט את כל הסיווגים. באותם מקרים שסיווגים אלו לא מתאימים ניתן	תיאוריית אבטיפוס

	ועצמים	להשתמש לסיווג מחלקה ע"י עצם טיפוס המייצג אותה. למשל: משחק יכול לייצג את המחלקה של משחקים למרות שאין שום תכונות משותפות לכל המשחקים.	
53	מערכת מחשב, חלקי		<u>תיעוד נהלים</u>
22,43		מקביל ל-HRS, IRS, SRS.	<u>תיק אפיון</u>
43		אין לו מקבילה ב-498 שמסתיים בסוף פיתוח הפרוייקט.	<u>תיק מערכת תחזוקה</u>
43		מקביל ל-STR, STD, STP.	<u>תיק סיכום מבדקים</u>
43		מקביל ל-IDD, SDD, DBDD.	<u>תיק עיצוב</u>
14			<u>תכן</u>
76	תכן תוכנה תכן מבנה נתונים תכן תהליכים אי-תלות פונקציונלית מודולריות הפשטה הסתרת מידע	תפקידו לקבוע את התמונה הכוללת של המודולים המרכיבים את המע', כולל הבקרה והמנשקים ביניהם. חלוקת מע' למודולים מתבצעת בהסתמך על עקרונות התכן של מודולריות, הפשטה והסתרת מידע שמהם נובע המושג של אי-תלות פונקציונלית.	<u>תכן ארכיטקטורה (מודולים)</u>
75	תכן תוכנה תכן תהליכים תכן ארכיטקטורה הפשטה הסתרת מידע מודולריות	עקרונות התכן: 1. מודולריות. 2. הפשטה. 3. הסתרת מידע.	<u>תכן מבנה/נתונים</u>
75,79		ראה Data Flow Oriented.	<u>תכן מונחה זרימת נתונים</u>
,75 ,82 ,83 ,84 ,85 ,86 ,87 89, 88	תכן תוכנה JSD LCP DSSD תרשים מדרגי/מבני	ישים היטב במע' להן מבנה מידע מדרגי מוגדר היטב. לדוגמה: מע' למנהל עסקים, מע' הפעלה ויישומי CAD/CAM/CAE/CAI. המטרה של תכן זה היא ליצור תיאור תפקודי של המע'. המודולים נגזרים (כמוצר לוואי byproduct) של התהליכים. ייצוג המע' הוא ע"י דיאגרמה מדרגית. שיטות תכן עיקריות: 1. פיתוח מע' עפ"י מבני נתונים – DSSD. 2. שיטת JSD. 3. בנייה לוגית של תוכניות LCP. המשותף לשיטות: 1. הערכת/בחירת מאפייני מבני נתונים. 2. ייצוג נתונים בצורות יסודיות, כגון: סדרה, בחירה וחזרה. 3. ייצוג מבנה הנתונים ממופה למדרג של בקרה. 4. עידון מדרגה התוכנה עפ"י כללים של השיטה. 5. בסוף מתקבל תיאור תפקודי של התוכנה.	<u>תכן מונחה מבנה נתונים</u>
75,90	תכן תוכנה	Object Oriented Design. עצם + מודול.	<u>תכן מונחה עצמים OOD</u>
76	תכן תוכנה תכן מבנה נתונים תכן ארכיטקטורה	מציג את הזרימה במערכת ("איך המע' פועלת"). תיאור התהליכים חייב להיות חד-משמעי ולכן חייבים להשתמש בכלים פורמליים.	<u>תכן תהליכים</u>
,74 77, 75	ארכיטקטורת תכן תכן הנתונים תכן הארכיטקטורה תכן תהליכים תכן מנשק המשתמש תכן מונחה זרימת נתונים תכן מונחה מבנה נתונים תכן מונחה עצמים תכן ע"י שפות מפרט פורמליות	תהליך באמצעותו מתרגמים דרישות לייצוג ע"י התוכנה. זהו הגרעין של הנדסת תוכנה. כל שלב של התהליך מהווה עידון של השלב הקודם לו. התכן הוא הבסיס עליו מושגת מימוש התוכנה. תרשים: יישום עם ובלי תכן – עמ' 74. כללי תכן טוב: 1. תכן צריך להציג ארגון מדרגי של מרכיבי תוכנה עם בקרה חכמה ביניהם. 2. תכן צריך להיות מודולרי, דהיינו מחולק לוגית עפ"י פונקציונליות. 3. תכן צריך להכיל ייצוג נתונים וייצוג תהליכים עם אבחנה ברורה ביניהם. 4. תכן צריך להוביל למודולים של פרוצדורות ושגרות המייצגים תפקודים בלתי תלויים. 5. תכן חייב להיות בר שחזור כנגזר ממפרט דרישות התוכנה. בתכן 2 שלבים ראשיים: 1. תכן על/מקדים (top level/preliminary) המייצג את ארכיטקטורת התוכנה והנתונים. 2. תכן מפורט (detailed) המעידן את הייצוג הארכיטקטוני לייצוג אלגוריתמי ולייצוג של מבני נתונים מפורטים. לתכן 4 מרכיבים: 1. תכן הנתונים. 2. תכן הארכיטקטורה (מודולים). 3. תכן התהליכים. 4. תכן מנשק המשתמש. תרשים – עמ' 77.	<u>תכן תוכנה יסודות התכן</u> <u>תכן</u>

		<p>מתודולוגיות לביצוע התכן:</p> <ol style="list-style-type: none"> 1. תכן מונחה זרימת נתונים. 2. תכן מונחה מבנה נתונים. 3. תכן מונחה עצמים. 4. תכן ע"י שפות מפרט פורמליות. <p>Good design is:</p> <ul style="list-style-type: none"> - Complete. - Economical. - Constructive. - Uniform. - Testable. 	
17	המודל הספירלי	הגדרת מטרות, חלופות ואילוצים.	<u>תכנון</u>
114		ראה קידוד.	<u>תרגום, תהליך</u>
89-84	תכן מונחה מבנה נתונים	השיטה הנפוצה ביותר לייצוג תוכנה היא הייצוג המדרגי דמוי עץ הנקרא תרשים מבני. דוגמה – עמ' 89-84.	<u>תרשים מדרגי</u> – <u>תרשים מבני</u>